## [136. 只出现一次的数字](https://leetcode-cn.com/problems/single-number/)

### 1 原题目

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

**示例 1:**

**输入:** [2,2,1]

**输出:** 1

**示例 2:**

**输入:** [4,1,2,1,2]

**输出:** 4

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

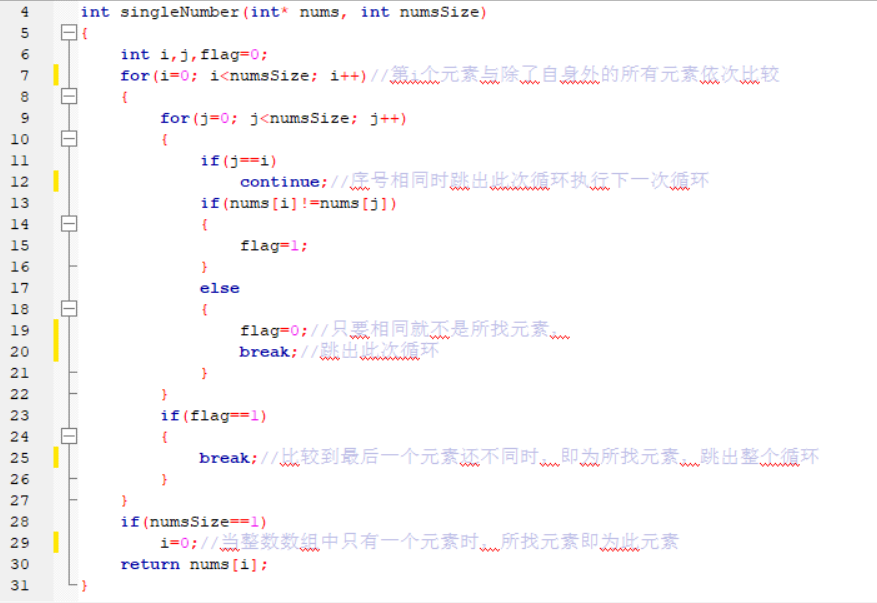
数学—异或运算。（解法二用到）

### 3 解法

#### 3.1 解法1（C语言已过）

通过数组内元素的比较来实现查找出只出现一次的那个元素。

本人代码：



#### 3.2 解法2（C语言已过）

用异或运算来找。

**异或运算介绍：**

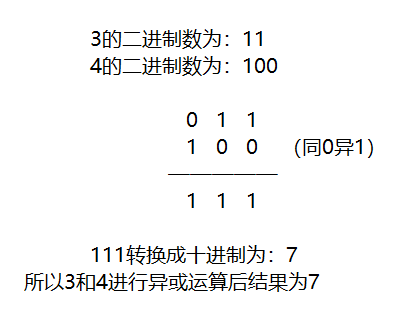
异或（xor）是一个数学运算符。它应用于逻辑运算。异或的数学符号为“⊕”，计算机符号为“xor”，在C语言里用  ^ 来表示。

运算法则：

将a、b两个数转换成二进制数。位数对其，二进制下用1表示真，0表示假。

则异或的运算法则为：0⊕0=0，1⊕0=1，0⊕1=1，1⊕1=0（同为0，异为1）。

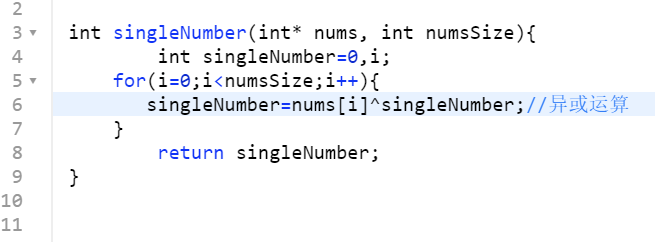
下图展示的是3和4两个十进制数经过异或运算得到的结果。



7和3进行异或运算结果为4。（运算方法如上图）

7和4进行异或运算结果为3。（运算方法如上图）

本人代码：



## [389. 找不同](https://leetcode-cn.com/problems/find-the-difference/)

### 1 原题目

给定两个字符串 ***s*** 和 ***t***，它们只包含小写字母。

字符串 **t** 由字符串 **s** 随机重排，然后在随机位置添加一个字母。

请找出在 ***t*** 中被添加的字母。

**示例:**

输入：

s = "abcd"

t = "abcde"

输出：

e

解释：

'e' 是那个被添加的字母。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

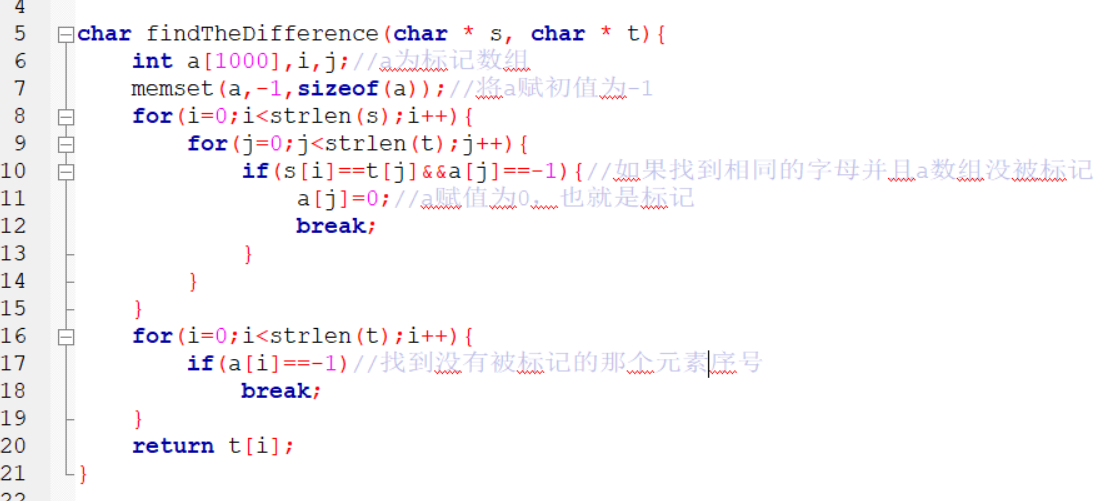
数学—异或运算。（解法二用到）

### 3 解法

#### 3.1 解法1（C语言已过）

定义一个整型数组作为标记数组，将字符串t和字符串s比较，如果相同并且数组未被标记，就将这个元素号标记上，最后找出未被标记的那个元素即为新添加字母。

本人代码：



#### 3.2 解法2（C语言已过）

将字母转换成整型然后进行异或运算来查找。

**异或运算介绍：**

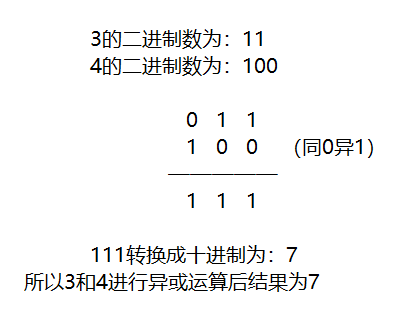
异或（xor）是一个数学运算符。它应用于逻辑运算。异或的数学符号为“⊕”，计算机符号为“xor”，在C语言里用  ^ 来表示。

运算法则：

将a、b两个数转换成二进制数。位数对其，二进制下用1表示真，0表示假。

则异或的运算法则为：0⊕0=0，1⊕0=1，0⊕1=1，1⊕1=0（同为0，异为1）。

下图展示的是3和4两个十进制数经过异或运算得到的结果。



7和3进行异或运算结果为4。（运算方法如上图）

7和4进行异或运算结果为3。（运算方法如上图）

本人代码：



## [1014. 最佳观光组合](https://leetcode-cn.com/problems/best-sightseeing-pair/)

### 1 原题目

给定正整数数组 A，A[i] 表示第 i 个观光景点的评分，并且两个景点 i 和 j 之间的距离为 j - i。

一对景点（i < j）组成的观光组合的得分为（A[i] + A[j] + i - j）：景点的评分之和减去它们两者之间的距离。

返回一对观光景点能取得的最高分。

**示例：**

输入：[8,1,5,2,6]

输出：11

解释：i = 0, j = 2, A[i] + A[j] + i - j = 8 + 5 + 0 - 2 = 11

**提示：**

2 <= A.length <= 50000

1 <= A[i] <= 1000

### 2 基础知识

数学（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

景点观光组合得分是A[i]+A[j]+i-j;(i < j)。题目告诉我们的这个i要小于j这个条件很重要。我们试想下，暴力的解法就是两遍for循环。对第一层的元素j在他之前寻找此时得分的最大值。在寻找的过程中，其实A[j]-j是不变的，我们找的就是A[i]+i的最大值。所以我们可以一遍for循环更新A[i]+i的最大值，并找出观光组合得分最高的分数。

本人代码：

|  |
| --- |
| class Solution {  public:      int maxScoreSightseeingPair(vector<int>& A) {          //mx记录从前到后A[i]+i的最大值          int mx = A[0],i,j,res = A[0];          //景点需满足i<j,每次景点的最大值就是找前面的A[i]+i的最大值          for(i = 1; i < A.size(); i++) {              res = max(res,mx + A[i] - i);              mx = max(mx,A[i] + i);          }          return res;      }  }; |

## [面试题56 - I. 数组中数字出现的次数](https://leetcode-cn.com/problems/shu-zu-zhong-shu-zi-chu-xian-de-ci-shu-lcof/)

### 1 原题目

一个整型数组 nums 里除两个数字之外，其他数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

**示例 1：**

输入：nums = [4,1,4,6]

输出：[1,6] 或 [6,1]

**示例 2：**

输入：nums = [1,2,10,4,1,4,3,3]

输出：[2,10] 或 [10,2]

**限制：**

2 <= nums <= 10000

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

位运算（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

因为题目告诉我们只有两个数字出现一次，其余所有数字都出现两次，所以我们可以利用位运算解题。但是异或运算算一遍的结果是那两个出现一次数字的异或结果。我们在异或结果的二进制中找到某一位上为1的那一位（因为异或运算结果为1代表两个数字的此位不相同）。然后利用此位对整个数组进行划分，首先，这两个不同的数字肯定分到不同组别，其余相同数字因为二进制表示一样所以也会分到同一组，所以将分成的两个组别进行异或运算就可以求出结果。

参考链接：<https://leetcode-cn.com/problems/shu-zu-zhong-shu-zi-chu-xian-de-ci-shu-lcof/solution/shu-zu-zhong-shu-zi-chu-xian-de-ci-shu-by-leetcode/>

本人代码：

|  |
| --- |
| class Solution {  public:      vector<int> singleNumbers(vector<int>& nums) {          int ret = 0;          for (int n : nums)              ret ^= n;//异或运算          int div = 1;          while ((div & ret) == 0)//与运算              div <<= 1;//左移（左移1位就是乘2）          int a = 0, b = 0;          for (int n : nums)              if (div & n)                  a ^= n;              else                  b ^= n;          return vector<int>{a, b};      }  }; |

## [13. 罗马数字转整数](https://leetcode-cn.com/problems/roman-to-integer/)

### 1 原题目

罗马数字包含以下七种字符: I， V， X， L，C，D 和 M。

字符 数值

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。 27 写做  XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。

X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。

C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。

**示例 1:**

输入: "III"

输出: 3

**示例 2:**

输入: "IV"

输出: 4

**示例 3:**

输入: "IX"

输出: 9

**示例 4:**

输入: "LVIII"

输出: 58

解释: L = 50, V= 5, III = 3.

**示例 5:**

输入: "MCMXCIV"

输出: 1994

解释: M = 1000, CM = 900, XC = 90, IV = 4.

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

新开辟一个整型数组，将所给的罗马数字一一转换成数字，因为存在6中特例，所以在编写程序时，要有一个判断大小的代码，以此来解决那特殊的6种情况，之后的实现加减的运算就从整型数组中的数字来运算。

本人代码：

int romanToInt(char \* s)

{

int a[1000]={},i,b=0;

for(i=0; i<strlen(s); i++)//将字符数组的字母转换成所对应的数字一一对应存储起来

{

if(s[i]=='I')

a[i]=1;

if(s[i]=='V')

a[i]=5;

if(s[i]=='X')

a[i]=10;

if(s[i]=='L')

a[i]=50;

if(s[i]=='C')

a[i]=100;

if(s[i]=='D')

a[i]=500;

if(s[i]=='M')

a[i]=1000;

}

for(i=0; i<strlen(s); i++)

{

if(a[i+1]>a[i]&&i!=strlen(s)-1)//后一个大的话就要后减前，注意不能判断到最后

{ //注意i不能等于strlen（s）-1，如果等于的话最后一个就没参加运算

b=a[i+1]-a[i]+b;

i++;//因为小于需要相减，所以两个字母参加了运算，所以需要跳过i后面的那一项

}

else

b=b+a[i];

}

return b;

}

## [38. 报数](https://leetcode-cn.com/problems/count-and-say/)

### 1 原题目

报数序列是一个整数序列，按照其中的整数的顺序进行报数，得到下一个数。其前五项如下：

1. 1

2. 11

3. 21

4. 1211

5. 111221

1 被读作  "one 1"  ("一个一") , 即 11。

11 被读作 "two 1s" ("两个一"）, 即 21。

21 被读作 "one 2",  "one 1" （"一个二" ,  "一个一") , 即 1211。

注意：整数顺序将表示为一个字符串。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

开辟一个静态二维数组来记录第n次报数的整串数字，由题意告诉我们数字的记录要用字符型，所以用b（字符型）记录这个数字，c（整型）来记录出现次数，并存储在数组中。

本人代码：

char \* countAndSay(int n){

static char a[32][5000],b;//开辟静态数组，如果不这样开辟，值无法返回

memset(a,0,sizeof(a));//数组的初始化

a[1][0]='1';

a[1][1]='\0';

a[2][0]='1';

a[2][1]='1';

a[2][2]='\0';

int i,k,m,j,c;

for(i=3;i<=n;i++){

m=0;

j=i-1;

for(k=0;k<strlen(a[j]);k++){

b=a[j][k];//b用来记录这个数字是什么

c=1;

while(a[j][k]==a[j][k+1]){//因为出现与前一次记录不同的数字就要重新记录，

//所以循环的条件是判断是否和后一个相同

c++;//c用来记录这个数字出现的次数

k++;

}

a[i][m++]=c+'0';//因为返回的是字符型，所以转换一下

a[i][m++]=b;

}

a[i][m++]='\0';

}

return a[n];

}

## [384. 打乱数组](https://leetcode-cn.com/problems/shuffle-an-array/)

### 1 原题目

打乱一个没有重复元素的数组。

**示例:**

// 以数字集合 1, 2 和 3 初始化数组。

int[] nums = {1,2,3};

Solution solution = new Solution(nums);

// 打乱数组 [1,2,3] 并返回结果。任何 [1,2,3]的排列返回的概率应该相同。

solution.shuffle();

// 重设数组到它的初始状态[1,2,3]。

solution.reset();

// 随机返回数组[1,2,3]打乱后的结果。

solution.shuffle();

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

暴力算法。（解法一用到）

洗牌算法。（解法二用到）

### 3 解法

#### 3.1 解法1（C语言已过）

随机排列的概率相同时是每次在数组长度内随机取数，取过了的就标记出来，直到取完。

本人代码：

typedef struct {

int \*arr;//存储数组的元素

int len;//存储数组的长度

} Solution;

Solution\* solutionCreate(int\* nums, int numsSize) {

Solution\* obj = (Solution\*)malloc(sizeof(Solution));

obj->len=numsSize;

if(numsSize>0){//只有当传进来的数组长度大于0时

//才将传进来的数组赋值给数组arr

obj->arr=nums;

}

return obj;

}

/\*\* Resets the array to its original configuration and return it. \*/

int\* solutionReset(Solution\* obj, int\* retSize) {

int \*nums=(int \*)malloc(1000\*sizeof(int));

memset(nums,0,sizeof(1000\*sizeof(int)));

if(obj->len>0){//只有当数组的长度大于0时

//才将它置为初始状态

nums=obj->arr;

}

\*retSize=obj->len;

return nums;

}

/\*\* Returns a random shuffling of the array. \*/

int\* solutionShuffle(Solution\* obj, int\* retSize) {

int i,j=0,k,array[1000]={0};//数组array为标记数组

int \*arr1=(int \*)malloc(1000\*sizeof(int));

memset(arr1,0,sizeof(1000\*sizeof(int)));

for(i=0;i<obj->len;i++){

if(j==0){//第一次随机取数要留着

//rand为随机取数的函数

k=rand()%obj->len;//随机取数之后在对数组长度求余是为了使随机数的范围落在

//[0——obj->len-1]这个范围内，与数组的下标对应

array[k]=1;

}

else{//从第二次开始要判断随机取出来的数字是否被标记过

//如果被标记了，重新随机取数

k=rand()%obj->len;

while(array[k]==1)

k=rand()%obj->len;

array[k]=1;

}

arr1[j++]=obj->arr[k];

}

\*retSize=obj->len;

return arr1;

}

void solutionFree(Solution\* obj) {

free(obj);

}

#### 3.2 解法2（C语言已过）

核心代码部分用洗牌算法来做，其余部分通过暴力算法的代码修改而来。

**洗牌算法之一介绍：**

****

截图来自**：**<https://blog.csdn.net/qq_26399665/article/details/79831490>（洗牌算法的更多介绍也在这里）

本人代码：

typedef struct {

int \*arr;//存储数组的元素

int ar[1000];//备份元素，存在这里的不做更改

int len;//存储数组的长度

} Solution;

Solution\* solutionCreate(int\* nums, int numsSize) {

Solution\* obj = (Solution\*)malloc(sizeof(Solution));

obj->len=numsSize;

if(numsSize>0){//只有当传进来的数组长度大于0时

//才将传进来的数组赋值给数组arr和ar

obj->arr=nums;

for(int i=0;i<numsSize;i++)

obj->ar[i]=nums[i];

}

return obj;

}

/\*\* Resets the array to its original configuration and return it. \*/

int\* solutionReset(Solution\* obj, int\* retSize) {

int \*nums=(int \*)malloc(1000\*sizeof(int));

memset(nums,0,sizeof(1000\*sizeof(int)));

if(obj->len>0){//只有当数组的长度大于0时

//才将它置为初始状态

for(int i=0;i<obj->len;i++)

nums[i]=obj->ar[i];

}

\*retSize=obj->len;

return nums;

}

/\*\* Returns a random shuffling of the array. \*/

int\* solutionShuffle(Solution\* obj, int\* retSize) {

int i,j=0,tmp,rnd;

int \*arr1=(int \*)malloc(1000\*sizeof(int));

memset(arr1,0,sizeof(1000\*sizeof(int)));

arr1=obj->arr;

for(i=0;i<obj->len;i++){

//rand为随机取数的函数

rnd=rand()%(obj->len-i)+i;这种写法可以实现在[i,obj->len-1]区间内的取随机数

tmp=arr1[i];//交换

arr1[i]=arr1[rnd];

arr1[rnd]=tmp;

}

\*retSize=obj->len;

return arr1;

}

void solutionFree(Solution\* obj) {

free(obj);

}

## [9. 回文数](https://leetcode-cn.com/problems/palindrome-number/)

### 1 原题目

判断一个整数是否是回文数。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。

**示例 1:**

输入: 121

输出: true

**示例 2:**

输入: -121

输出: false

解释: 从左向右读, 为 -121 。 从右向左读, 为 121- 。因此它不是一个回文数。

**示例 3:**

输入: 10

输出: false

解释: 从右向左读, 为 01 。因此它不是一个回文数。

**进阶:**

你能不将整数转为字符串来解决这个问题吗？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

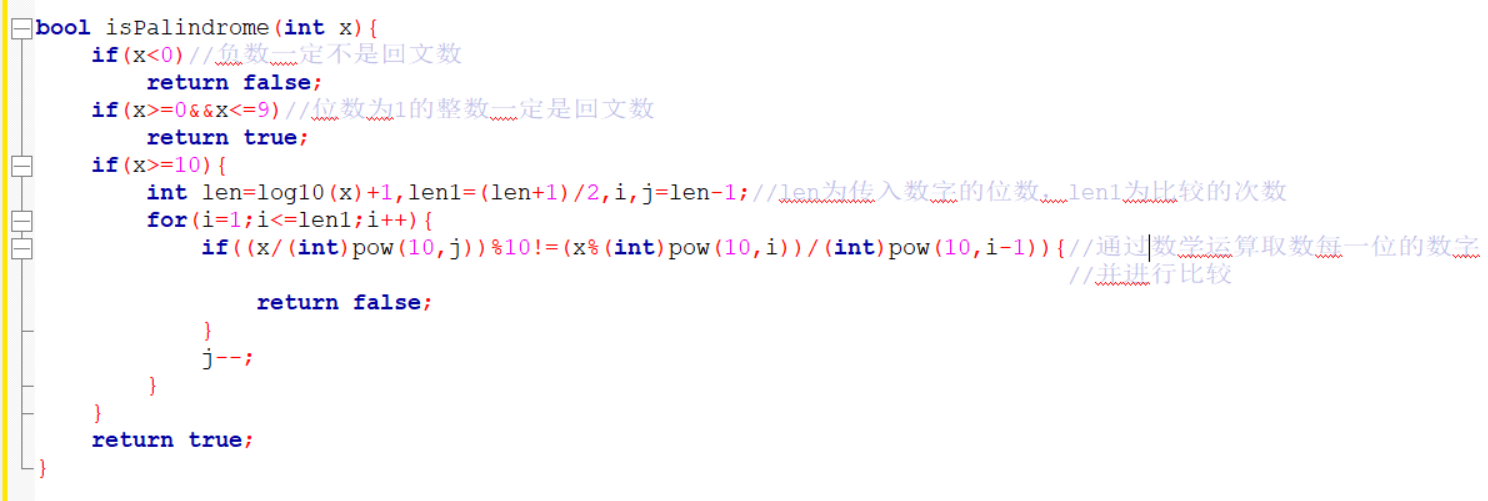
无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过把数字的每一位上的数字取出来进行比较来判断出来整数是否为回文数。

本人代码：



## [258. 各位相加](https://leetcode-cn.com/problems/add-digits/)

### 1 原题目

给定一个非负整数 num，反复将各个位上的数字相加，直到结果为一位数。

**示例:**

输入: 38

输出: 2

解释: 各位相加的过程为：3 + 8 = 11, 1 + 1 = 2。 由于 2 是一位数，所以返回 2。

**进阶:**

你可以不使用循环或者递归，且在 O(1) 时间复杂度内解决这个问题吗？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

有规律，规律如下：

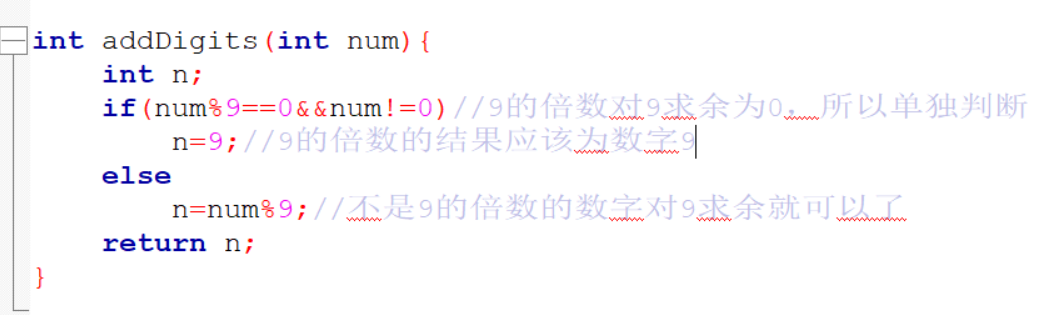
0～9的结果就是0～9

10～18的结果是1、2、3 …… 9

19～27的结果是1、2、3 …… 9

0是特列，从1开始就9个就是一个循环。

本人代码：



## [171. Excel表列序号](https://leetcode-cn.com/problems/excel-sheet-column-number/)

### 1 原题目

给定一个Excel表格中的列名称，返回其相应的列序号。

例如，

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

...

**示例 1:**

输入: "A"

输出: 1

**示例 2:**

输入: "AB"

输出: 28

**示例 3:**

输入: "ZY"

输出: 701

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

就是进制转换而已。

给你十进制数701=1\*10^0+0\*10^1+7\*10^2

理应用到Excel表格的序号上，ZY=26\*10^0+26\*10^1=701,

大家自己体会一下，本人也是百度了一下才豁然开朗的。

本人代码：

图片包含 屏幕截图

描述已自动生成

## [12. 整数转罗马数字](https://leetcode-cn.com/problems/integer-to-roman/)

### 1 原题目

罗马数字包含以下七种字符： I， V， X， L，C，D 和 M。

字符 数值

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。 27 写做  XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。

X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。

C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个整数，将其转为罗马数字。输入确保在 1 到 3999 的范围内。

**示例 1:**

输入: 3

输出: "III"

**示例 2:**

输入: 4

输出: "IV"

**示例 3:**

输入: 9

输出: "IX"

**示例 4:**

输入: 58

输出: "LVIII"

解释: L = 50, V = 5, III = 3.

**示例 5:**

输入: 1994

输出: "MCMXCIV"

解释: M = 1000, CM = 900, XC = 90, IV = 4.

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

列出所有情况，用if语句来判断。

本人代码：

char \* intToRoman(int num){

char \*arr=(char \*)malloc(1000\*sizeof(char));

memset(arr,0,1000\*sizeof(arr[0]));

int len=log10(num)+1,i,num1,j,num2,k=0,len1=len-1;

for(i=0;i<len;i++){

num1=((num/(int)pow(10,len-1-i))%10)\*(int)pow(10,len1--);//算出整数的每一位对应的数

//如果num是425，那么第一次mun1=400,第二次num1=20,第三次num1=5

if(num1>=1000){

num2=num1/1000;

for(j=1;j<=num2;j++)

arr[k++]='M';

}

if(num1==900){

arr[k++]='C';

arr[k++]='M';

}

if(num1>=500&&num1<900){

arr[k++]='D';

num2=(num1-500)/100;

for(j=1;j<=num2;j++)

arr[k++]='C';

}

if(num1==400){

arr[k++]='C';

arr[k++]='D';

}

if(num1<400&&num1>=100){

num2=num1/100;

for(j=1;j<=num2;j++)

arr[k++]='C';

}

if(num1==90){

arr[k++]='X';

arr[k++]='C';

}

if(num1>=50&&num1<90){

arr[k++]='L';

num2=(num1-50)/10;

for(j=1;j<=num2;j++)

arr[k++]='X';

}

if(num1==40){

arr[k++]='X';

arr[k++]='L';

}

if(num1<40&&num1>=10){

num2=num1/10;

for(j=1;j<=num2;j++)

arr[k++]='X';

}

if(num1==9){

arr[k++]='I';

arr[k++]='X';

}

if(num1>=5&&num1<9){

arr[k++]='V';

num2=(num1-5)/1;

for(j=1;j<=num2;j++)

arr[k++]='I';

}

if(num1==4){

arr[k++]='I';

arr[k++]='V';

}

if(num1<4){

num2=num1/1;

for(j=1;j<=num2;j++)

arr[k++]='I';

}

}

return arr;

}

## [20. 有效的括号](https://leetcode-cn.com/problems/valid-parentheses/)

### 1 原题目

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

**示例 1:**

输入: "()"

输出: true

**示例 2:**

输入: "()[]{}"

输出: true

**示例 3:**

输入: "(]"

输出: false

**示例 4:**

输入: "([)]"

输出: false

**示例 5:**

输入: "{[]}"

输出: true

### 2 基础知识

#### 2.1数据结构

栈（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

开辟一个字符数组str，将pop指针指向栈底0的位置，然后遍历所传入的字符串，如果是三种括号的左括号就存入栈中，否则将当前括号与栈顶的括号进行匹配，如果匹配上就继续，匹配不上就说明括号是错误的。遍历结束后还要判断pop指针是否为0，不是0说明栈中还有括号没进行匹配，也就是错误的，如果为0，说明都匹配了，说明是正确的。

本人代码：

bool isValid(char \* s){

int len=strlen(s),i,pop=0;//len为字符串的长度，pop是栈的指针

if(len%2!=0)//长度是奇数时是错误的

return false;

if(len==0)//长度是0时是正确的

return true;

char str[10000];//栈

memset(str,0,sizeof(str));//初始化数组

for(i=0;i<len;i++){

if(s[i]=='('||s[i]=='['||s[i]=='{'){//如果是这三种就进栈

str[pop++]=s[i];

}

else{

if(pop>=1)//当栈中存入了的时候才将指针减一

pop=pop-1;

//括号匹配

if((s[i]==']'&&str[pop]=='[')||(s[i]=='}'&&str[pop]=='{')||(s[i]==')'&&str[pop]=='(')){

//匹配正确就将栈顶元素取出，因为上面有一步pop=pop-1，所以此时这里不用进行额外的操作

}

else

return false;//匹配不上就是错误

}

}

if(pop==0)

return true;//最后栈中无元素时，才是正确的

return false;

}

## [32. 最长有效括号](https://leetcode-cn.com/problems/longest-valid-parentheses/)

### 1 原题目

给定一个只包含 '(' 和 ')' 的字符串，找出最长的包含有效括号的子串的长度。

**示例 1:**

输入: "(()"

输出: 2

解释: 最长有效括号子串为 "()"

**示例 2:**

输入: ")()())"

输出: 4

解释: 最长有效括号子串为 "()()"

### 2 基础知识

栈（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

具体做法是我们始终保持栈底元素为当前已经遍历过的元素中「最后一个没有被匹配的右括号的下标」，这样的做法主要是考虑了边界条件的处理，栈里其他元素维护左括号的下标：

对于遇到的每个‘(’ ，我们将它的下标放入栈中

对于遇到的每个‘)’ ，我们先弹出栈顶元素表示匹配了当前右括号：

如果栈为空，说明当前的右括号为没有被匹配的右括号，我们将其下标放入栈中来更新我们之前提到的「最后一个没有被匹配的右括号的下标」

如果栈不为空，当前右括号的下标减去栈顶元素即为「以该右括号为结尾的最长有效括号的长度」

我们从前往后遍历字符串并更新答案即可。

需要注意的是，如果一开始栈为空，第一个字符为左括号的时候我们会将其放入栈中，这样就不满足提及的「最后一个没有被匹配的右括号的下标」，为了保持统一，我们在一开始的时候往栈中放入一个值为 -1−1 的元素。

参考链接：<https://leetcode-cn.com/problems/longest-valid-parentheses/solution/zui-chang-you-xiao-gua-hao-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      int longestValidParentheses(string s) {          int res = 0;          stack<int> left;          //先加入-1是为了避免一开始是左括号时，没有最后一个没被匹配的右括号的下标          left.push(-1);          for(int i = 0; i < s.length(); i++) {              if(s[i] == '(') {//遇见左括号                  left.push(i);//下标存入栈中              }              else {//遇见右括号                  left.pop();//弹出左括号代表匹配了右括号                  if(left.empty()) {//栈为空                      left.push(i);//加入下标代表最后一个没被匹配的右括号的下标                      continue;                  }                  //栈不为空，当前右括号的下标减去栈顶元素即为以该右括号为结尾的最长有效括号的长度                  res = max(res,i - left.top());              }          }          return res;      }  }; |

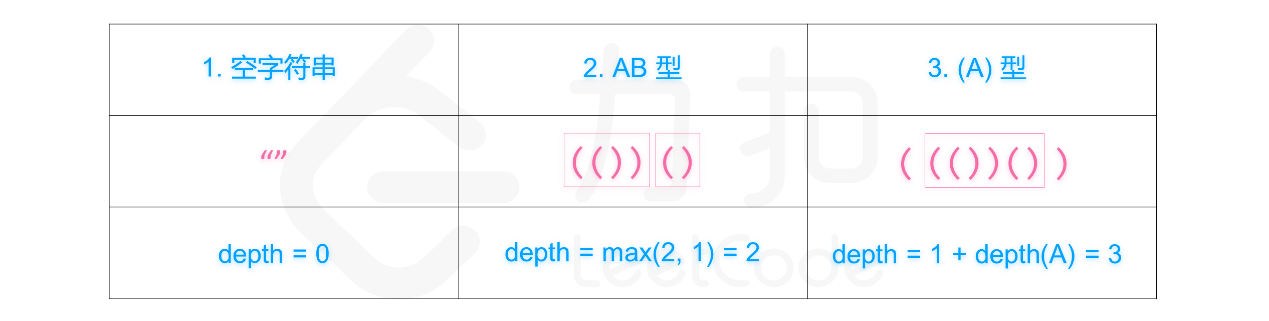
## [1111. 有效括号的嵌套深度](https://leetcode-cn.com/problems/maximum-nesting-depth-of-two-valid-parentheses-strings/)

### 1 原题目

有效括号字符串 定义：对于每个左括号，都能找到与之对应的右括号，反之亦然。详情参见题末「有效括号字符串」部分。

嵌套深度 depth 定义：即有效括号字符串嵌套的层数，depth(A) 表示有效括号字符串 A 的嵌套深度。详情参见题末「嵌套深度」部分。

有效括号字符串类型与对应的嵌套深度计算方法如下图所示：



给你一个「有效括号字符串」 seq，请你将其分成两个不相交的有效括号字符串，A 和 B，并使这两个字符串的深度最小。

不相交：每个 seq[i] 只能分给 A 和 B 二者中的一个，不能既属于 A 也属于 B 。

A 或 B 中的元素在原字符串中可以不连续。

A.length + B.length = seq.length

深度最小：max(depth(A), depth(B)) 的可能取值最小。

划分方案用一个长度为 seq.length 的答案数组 answer 表示，编码规则如下：

answer[i] = 0，seq[i] 分给 A 。

answer[i] = 1，seq[i] 分给 B 。

如果存在多个满足要求的答案，只需返回其中任意 一个 即可。

**示例 1：**

输入：seq = "(()())"

输出：[0,1,1,1,1,0]

**示例 2：**

输入：seq = "()(())()"

输出：[0,0,0,1,1,0,1,1]

解释：本示例答案不唯一。

按此输出 A = "()()", B = "()()", max(depth(A), depth(B)) = 1，它们的深度最小。

像 [1,1,1,0,0,1,1,1]，也是正确结果，其中 A = "()()()", B = "()", max(depth(A), depth(B)) = 1 。

**提示：**

1 < seq.size <= 10000

**有效括号字符串：**

仅由 "(" 和 ")" 构成的字符串，对于每个左括号，都能找到与之对应的右括号，反之亦然。

下述几种情况同样属于有效括号字符串：

1. 空字符串

2. 连接，可以记作 AB（A 与 B 连接），其中 A 和 B 都是有效括号字符串

3. 嵌套，可以记作 (A)，其中 A 是有效括号字符串

**嵌套深度：**

类似地，我们可以定义任意有效括号字符串 s 的 嵌套深度 depth(S)：

1. s 为空时，depth("") = 0

2. s 为 A 与 B 连接时，depth(A + B) = max(depth(A), depth(B))，其中 A 和 B 都是有效括号字符串

3. s 为嵌套情况，depth("(" + A + ")") = 1 + depth(A)，其中 A 是有效括号字符串

例如：""，"()()"，和 "()(()())" 都是有效括号字符串，嵌套深度分别为 0，1，2，而 ")(" 和 "(()" 都不是有效括号字符串。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://leetcode-cn.com/problems/maximum-nesting-depth-of-two-valid-parentheses-strings/solution/you-xiao-gua-hao-de-qian-tao-shen-du-by-leetcode-s/>

本人代码：

class Solution {

public:

    vector<int> maxDepthAfterSplit(string seq) {

        vector<int>ans;

        int d = 0;//记录左括号出现的次数（即嵌套深度）

        for(char c : seq) {

            if(c == '(') {

                d++;//左括号d加1

                ans.push\_back(d % 2);

            }

            else {

                ans.push\_back(d % 2);

                d--;//右括号会匹配掉一个左括号d减1

            }

        }

        return ans;

    }

};

## [28. 实现strStr()](https://leetcode-cn.com/problems/implement-strstr/)

### 1 原题目

实现 strStr() 函数。

给定一个 haystack 字符串和一个 needle 字符串，在 haystack 字符串中找出 needle 字符串出现的第一个位置 (从0开始)。如果不存在，则返回  -1。

**示例 1:**

输入: haystack = "hello", needle = "ll"

输出: 2

**示例 2:**

输入: haystack = "aaaaa", needle = "bba"

输出: -1

**说明:**

当 needle 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当 needle 是空字符串时我们应当返回 0 。这与C语言的 strstr() 以及 Java的 indexOf() 定义相符。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

BF算法（解法一用到）

KMP算法（解法二用到）

### 3 解法

#### 3.1 解法1（C语言已过）

BF算法的基础上做了一点很小的优化。

BF算法详见百度百科：

<https://baike.baidu.com/item/BF%E7%AE%97%E6%B3%95/4326761?fr=aladdin>

或者自行搜索。

本人代码：

int strStr(char \* haystack, char \* needle){//BF算法

int h\_len=strlen(haystack),n\_len=strlen(needle),j=0,sign=-1,k;

if(n\_len==0)//短串长度为0时肯定是任何字符串的子串，所以返回0

return 0;

//就在for循环的判断条件这里做了优化，长串中后面的字符就不用每一个都重新开始比较，直接不比较

//因为后面的长度都小于短串，肯定匹配不上

for(k=0;k<=h\_len-n\_len;k++){

if(haystack[k]==needle[j]){

//记录这个首次相等时k的值，如果匹配失败，k的值要还原

sign=k;//匹配如果成功，直接返回这个记录的k值

j++;

k++;

while(haystack[k]==needle[j]&&j<n\_len&&k<h\_len){//注意不能超出两个数组的长度

j++;

k++;

}

if(j>=n\_len)//如果while循环结束j等于它的长度，就说明匹配成功了

return sign;

else{//当匹配失败时

k=sign;//k恢复原来的值，因为for循环，k还要加一次，就达到了从长串下一个字母开始重新比较的目的

sign=-1;

j=0;//j变为0，以便下次判断

}

}

}

return sign;

}

#### 3.2 解法2（C语言已过）

KMP算法，本人讲不明白，但是推荐一篇博客，我是看的这篇博客明白了KMP算法。

链接：<https://blog.csdn.net/starstar1992/article/details/54913261>

本人代码：

void pre\_next(char \*str,int \*next,int len){

int k=-1,i;

next[0]=-1;//next[0]初始化为-1，-1表示不存在相同的最大前缀和最大后缀

for(i=1;i<len;i++){

while(k>-1&&str[k+1]!=str[i])//如果下一个不同，那么k就变成next[k]，注意next[k]是小于k的，无论k取任何值。

k=next[k];//继续往前判断

if(str[k+1]==str[i])

k=k+1;

next[i]=k;//这个是把算的k的值（就是相同的最大前缀和最大后缀长）赋给next[q]

//如果next[i]=0，说明存在一个长度的相同的最大前缀和最大后缀，以此类推

}

}

int strStr(char \* haystack, char \* needle){

int h\_len=strlen(haystack),n\_len=strlen(needle),i,k=-1;

if(n\_len==0)//短串长度为0时肯定是任何字符串的子串，所以返回0

return 0;

int\* next=(int \*)malloc(100000\*sizeof(int));

memset(next,0,100000\*sizeof(next[0]));//数组初始化

pre\_next(needle,next,n\_len);//对短串进行预处理

for(i=0;i<h\_len;i++){

while(k>-1&&needle[k+1]!=haystack[i])

k=next[k];

if(needle[k+1]==haystack[i])

k=k+1;

if(k==n\_len-1)//说明k移动到短串的最末端

return i-n\_len+1;

}

return -1;

}

## [214. 最短回文串](https://leetcode-cn.com/problems/shortest-palindrome/)

### 1 原题目

给定一个字符串 s，你可以通过在字符串前面添加字符将其转换为回文串。找到并返回可以用这种方式转换的最短回文串。

**示例 1:**

输入: "aacecaaa"

输出: "aaacecaaa"

**示例 2:**

输入: "abcd"

输出: "dcbabcd"

### 2 基础知识

KMP算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://leetcode-cn.com/problems/shortest-palindrome/solution/shou-hua-tu-jie-cong-jian-dan-de-bao-li-fa-xiang-d/>

官方题解几里哇啦说了一大堆，我看不懂（惭愧），上面的说的比较通俗易懂。这道题暴力解法会超时，暴力解法就是把字符串后面的字符一个个加到原串前面并判断是否为回文串。所以我们可以从暴力解法推出来这个利用KMP算法的next数组的这个解法。直接将原串s翻转成rev\_s,然后加到s后面构建成新的字符串（中间加一个字符#，防止计算公共前后缀出错）。并求解出新字符串的最长公共前后缀，然后在rev\_s的后缀中砍掉这个前缀，再加到s前面就为答案。

本人代码：

|  |
| --- |
| class Solution {  public:      //求解next数组      //next[i] + 1表示字符串s的最长公共前后缀      void pre\_next(string& s,vector<int>& next) {          int k = -1;          for (int i = 1; i < s.size(); i++) {               while (k > -1 && s[k + 1] != s[i])                  k = next[k];               if (s[k + 1] == s[i])                  k++;              next[i] = k;          }      }      string shortestPalindrome(string s) {          string res = "";          string rev\_s = s;          reverse(rev\_s.begin(),rev\_s.end());          string new\_s = s + '#' + rev\_s;          vector<int> next(new\_s.size(),-1);          pre\_next(new\_s,next);          //结果就是求s的最长回文前缀，然后在rev\_s的后缀中砍掉这个前缀，再加到s前面。          res = rev\_s.substr(0,rev\_s.size() - (next[new\_s.size() - 1] + 1)) + s;          return res;      }  }; |

## [1392. 最长快乐前缀](https://leetcode-cn.com/problems/longest-happy-prefix/)

### 1 原题目

「快乐前缀」是在原字符串中既是 非空 前缀也是后缀（不包括原字符串自身）的字符串。

给你一个字符串 s，请你返回它的 最长快乐前缀。

如果不存在满足题意的前缀，则返回一个空字符串。

**示例 1：**

输入：s = "level"

输出："l"

解释：不包括 s 自己，一共有 4 个前缀（"l", "le", "lev", "leve"）和 4 个后缀（"l", "el", "vel", "evel"）。最长的既是前缀也是后缀的字符串是 "l" 。

**示例 2：**

输入：s = "ababab"

输出："abab"

解释："abab" 是最长的既是前缀也是后缀的字符串。题目允许前后缀在原字符串中重叠。

**示例 3：**

输入：s = "leetcodeleet"

输出："leet"

**示例 4：**

输入：s = "a"

输出：""

**提示：**

1 <= s.length <= 10^5

s 只含有小写英文字母

### 2 基础知识

KMP算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

我们在使用KMP算法时一般都会对一个数组next进行预处理，其实这个预处理的过程就是此题的求解。

next[i]表示关于字符串s的子串 s[0..i] 最长的既是前缀也是后缀的字符串(允许前后缀在原字符串中重叠)。但这个字符串不能是 s[0..i] 本身。例如当字符串为 s = abacaba 时，aba 就是最长的既是前缀也是后缀的字符串，next[6] 的值为 2（注意字符串的下标是从 0 开始的），s 的前缀 s[0..2] 和后缀 s[4..6] 均为 aba。

本人代码：

|  |
| --- |
| class Solution {  public:      string longestPrefix(string s) {          vector<int> next(s.length(),-1);          int k = -1,i;          /\*              next[i]表示关于字符串s的子串 s[0..i] 最长的既是前缀也是后缀的字符串(允许前后缀在原字符串中重叠)              但这个字符串不能是 s[0..i] 本身。例如当字符串为 s = abacaba 时，              aba 就是最长的既是前缀也是后缀的字符串，next[6] 的值为 2（注意字符串的下标是从 0 开始的）              s 的前缀 s[0..2] 和后缀 s[4..6] 均为 aba。          \*/          for(i = 1; i < s.length(); i++)//就是KMP算法中预处理的那部分函数          {              while(k > -1 && s[k + 1] != s[i])                  k = next[k];              if(s[k + 1] == s[i])                  k++;              next[i] = k;          }          return s.substr(0,next[s.length() - 1] + 1);      }  }; |

## [238. 除自身以外数组的乘积](https://leetcode-cn.com/problems/product-of-array-except-self/)

### 1 原题目

给定长度为 n 的整数数组 nums，其中 n > 1，返回输出数组 output ，其中 output[i] 等于 nums 中除 nums[i] 之外其余各元素的乘积。

**示例:**

输入: [1,2,3,4]

输出: [24,12,8,6]

**说明:** 请不要使用除法，且在 O(n) 时间复杂度内完成此题。

**进阶：**

你可以在常数空间复杂度内完成这个题目吗？（ 出于对空间复杂度分析的目的，输出数组不被视为额外空间。）

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

网上看的别人的思路，乘积=当前数左边的乘积\*当前数右边的乘积

本人代码：

图片包含 屏幕截图

描述已自动生成

## [397. 整数替换](https://leetcode-cn.com/problems/integer-replacement/)

### 1 原题目

给定一个正整数 n，你可以做如下操作：

1. 如果 n 是偶数，则用 n / 2替换 n。

2. 如果 n 是奇数，则可以用 n + 1或n - 1替换 n。

n 变为 1 所需的最小替换次数是多少？

**示例 1:**

输入:

8

输出:

3

解释:

8 -> 4 -> 2 -> 1

**示例 2:**

输入:

7

输出:

4

解释:

7 -> 8 -> 4 -> 2 -> 1

或

7 -> 6 -> 3 -> 2 -> 1

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

偶数只有一种变换，没什么好说的，问题就在奇数上，比如说给你一个奇数15，如果使它加1，那么可以连续两次除2的运算，而对它减1之后，只能进行一次除2的运算，所以奇数的时候要判断一下，如果这个数字加1能被4整除，那就使它加1，如果减1能被4整除，那就减一，但是要注意一个奇数，这个数字是3，对他一定要进行减1的操作，而不是加。

本人代码：

int integerReplacement(int n){

int count=0;

long long int m=n;//中间防止超出int型的范围，所以用长整型

while(m!=1){

if(m%2==0){//如果是偶数，就除2

m=m/2;

count++;

}

else{

if((m+1)%4==0&&m!=3){//要把3排除出去

m=m+1;

count++;

}

else{

m=m-1;

count++;

}

}

}

return count;

}

## [50. Pow(x, n)](https://leetcode-cn.com/problems/powx-n/)

### 1 原题目

实现 pow(x, n) ，即计算 x 的 n 次幂函数。

**示例 1:**

输入: 2.00000, 10

输出: 1024.00000

**示例 2:**

输入: 2.10000, 3

输出: 9.26100

**示例 3:**

输入: 2.00000, -2

输出: 0.25000

解释: 2-2 = 1/22 = 1/4 = 0.25

**说明:**

-100.0 < x < 100.0

n 是 32 位有符号整数，其数值范围是 [−231, 231 − 1] 。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

快速幂算法（解法一用到）

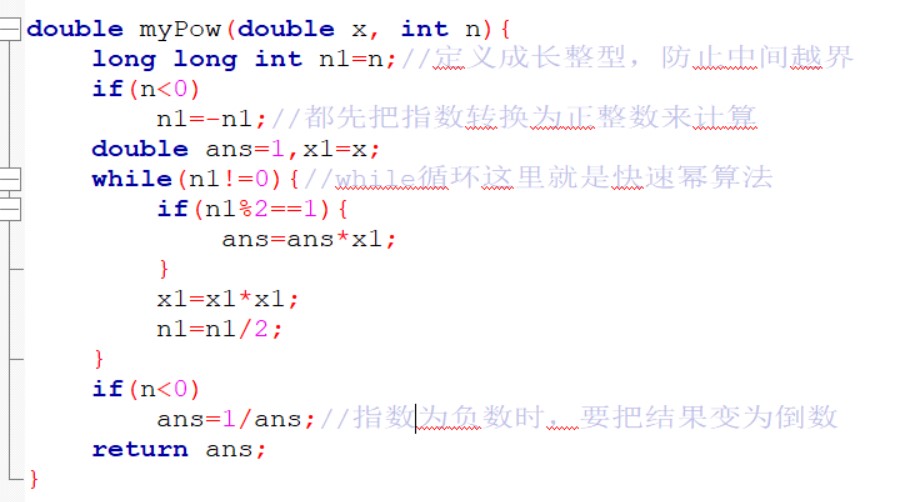
### 3 解法

#### 3.1 解法1（C语言已过）

利用快速幂算法来实现这个函数的功能。函数编写时要注意些细节问题

快速幂算法的讲解：<https://blog.csdn.net/u012961561/article/details/45422193>

本人代码：



## [155. 最小栈](https://leetcode-cn.com/problems/min-stack/)

### 1 原题目

设计一个支持 push，pop，top 操作，并能在常数时间内检索到最小元素的栈。

push(x) -- 将元素 x 推入栈中。

pop() -- 删除栈顶的元素。

top() -- 获取栈顶元素。

getMin() -- 检索栈中的最小元素。

**示例:**

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); --> 返回 -3.

minStack.pop();

minStack.top(); --> 返回 0.

minStack.getMin(); --> 返回 -2.

### 2 基础知识

#### 2.1数据结构

栈（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

栈的基本操作的实现，其中这个getMin()函数的实现我借助了辅助栈（就是在搞一个栈）。就是传进来的元素和辅助栈中的栈顶元素进行比较，如果小于等于栈顶元素，就进到辅助栈中，当原来的栈中要删除栈顶元素时，将这个要删除的栈顶元素和辅助栈的栈顶元素进行比较，如果相同，则辅助栈的栈顶元素也删除，不相同则辅助栈维持原样。

本人代码：

typedef struct {

int \*arr;//栈

int top;//栈的指针

int \*arr1;//辅助栈

int top1;//辅助栈的指针

} MinStack;

/\*\* initialize your data structure here. \*/

MinStack\* minStackCreate() {

MinStack \*obj=(MinStack \*)malloc(sizeof(MinStack));//开辟一个栈

obj->top=0;

obj->top1=1;

obj->arr=(int \*)malloc(10000\*sizeof(int));//开辟栈中的数组

obj->arr1=(int \*)malloc(10000\*sizeof(int));

memset(obj->arr,0,10000\*sizeof(int));//栈的数组的初始化

memset(obj->arr1,0,10000\*sizeof(int));

return obj;

}

void minStackPush(MinStack\* obj, int x) {//压栈

if(obj->top==0){//第一个传进来的元素要进到辅助栈中，方便后来的比较

obj->arr1[obj->top1]=x;

obj->top1++;

}

if(obj->top!=0&&x<=obj->arr1[obj->top1-1]){//比辅助栈的栈顶元素小于或等于就进到辅助栈中

obj->arr1[obj->top1]=x;

obj->top1++;

}

obj->arr[obj->top]=x;

obj->top++;

}

void minStackPop(MinStack\* obj) {//删除栈顶元素

if(obj->arr[obj->top-1]==obj->arr1[obj->top1-1])//如果要删除的这个元素等于辅助栈的栈顶元素

obj->top1=obj->top1-1; //则辅助栈的栈顶元素也删除

obj->top=obj->top-1;

}

int minStackTop(MinStack\* obj) {//取栈顶元素的值

int a=obj->arr[obj->top-1];

return a;

}

int minStackGetMin(MinStack\* obj) {//获取栈中的最小值

int min=obj->arr1[obj->top1-1];//辅助栈的栈顶元素就是栈中的最小值

return min;

}

void minStackFree(MinStack\* obj) {//释放栈

free(obj);

}

## [LCP 13. 寻宝](https://leetcode-cn.com/problems/xun-bao/)

### 1 原题目

我们得到了一副藏宝图，藏宝图显示，在一个迷宫中存在着未被世人发现的宝藏。

迷宫是一个二维矩阵，用一个字符串数组表示。它标识了唯一的入口（用 'S' 表示），和唯一的宝藏地点（用 'T' 表示）。但是，宝藏被一些隐蔽的机关保护了起来。在地图上有若干个机关点（用 'M' 表示），只有所有机关均被触发，才可以拿到宝藏。

要保持机关的触发，需要把一个重石放在上面。迷宫中有若干个石堆（用 'O' 表示），每个石堆都有无限个足够触发机关的重石。但是由于石头太重，我们一次只能搬一个石头到指定地点。

迷宫中同样有一些墙壁（用 '#' 表示），我们不能走入墙壁。剩余的都是可随意通行的点（用 '.' 表示）。石堆、机关、起点和终点（无论是否能拿到宝藏）也是可以通行的。

我们每步可以选择向上/向下/向左/向右移动一格，并且不能移出迷宫。搬起石头和放下石头不算步数。那么，从起点开始，我们最少需要多少步才能最后拿到宝藏呢？如果无法拿到宝藏，返回 -1 。

**示例 1：**

输入： ["S#O", "M..", "M.T"]

输出：16

解释：最优路线为： S->O, cost = 4, 去搬石头 O->第二行的M, cost = 3, M机关触发 第二行的M->O, cost = 3, 我们需要继续回去 O 搬石头。 O->第三行的M, cost = 4, 此时所有机关均触发 第三行的M->T, cost = 2，去T点拿宝藏。 总步数为16。

**示例 2：**

输入： ["S#O", "M.#", "M.T"]

输出：-1

解释：我们无法搬到石头触发机关

**示例 3：**

输入： ["S#O", "M.T", "M.."]

输出：17

解释：注意终点也是可以通行的。

**限制：**

1 <= maze.length <= 100

1 <= maze[i].length <= 100

maze[i].length == maze[j].length

S 和 T 有且只有一个

0 <= M的数量 <= 16

0 <= O的数量 <= 40，题目保证当迷宫中存在 M 时，一定存在至少一个 O 。

### 2 基础知识

状态压缩动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://leetcode-cn.com/problems/xun-bao/solution/xun-bao-bfs-dp-by-leetcode-solution/>

以下内容来自<https://leetcode-cn.com/u/yuan-zhu-yue-hua/> 此人在官方题解下的评论

* 主要思路
* 首先明确，要从起点到终点，必须触发所有的机关，且在去每个机关之前都要先去石头，即：
* 起点 -> 某堆石头 -> 机关1
* 机关1 -> 某堆石头 -> 机关2
* ...
* 机关n-1 -> 某堆石头 -> 机关n
* 机关n -> 终点
* 而这其中，去哪堆石头有多种选择，机关的顺序也有多种选择
* 但当我们确定了石头和机关的选择后，要求的其实就是上面的 -> 的最短路径了
* 而因为寻宝图的不变的，
* 所以确定了石头和机关的选择后，-> 的最短路径就是确定的了
* -> 连接的无非就是：
* 起点和石头、石头和机关、石头和终点
* 于是我们可以先预处理出 这些最短路径
* 进一步，我们计算上面图中每一行的最短路径，即
* 起点 -> 机关
* 机关 -> 机关
* 机关 -> 终点
* 即我们在预处理时，提前将石头给遍历了，找到了每一行中
* 经过哪块石头可以取得最短的路径
* 主要计算过程：
* 1. 因为有障碍，所以我们先 bfs 算出从起点到所有点的最短距离
* 如果不可达则为 -1
* 2. 对于每个机关，bfs 计算它到所有点的距离
* 那么此时 机关 -> 终点 就有了，因为这是不用经过石头的
* 且 机关 -> 石头 及 石头 -> 机关 也有了
* 反一下最短距离是不变的
* 3. 有了2，
* 起点 -> 某堆石头 -> 机关i 最短距离
* 只需固定 i ，遍历石头即可
* 且需要保证 起点 和 i 到当前遍历的 石头 是可达的
* 再遍历所有 i ，起点 -> 机关 就有了
* 机关i -> 某堆石头 -> 机关j 最短距离
* 只需固定 i 和 j ，遍历石头即可
* 且需要保证 i 和 j 到当前遍历的 石头 是可达的
* 再遍历所有 i 和 j，机关 -> 机关 就有了
* 预处理工作完成
* 我们将预处理结果保存在 dist 数组中
* 设 nb 为机关的个数
* 从某个机关出发，所以一共有 nb 行
* 到所有机关 + 起点 + 终点，所以一共 nb+2 列
* dist[i][nb+1] 表示从 机关i 到 终点 的最短距离
* dist[i][nb] 表示从 机关i 到 起点 的最短距离
* dist[i][j] 表示从 机关i 到 机关j 的最短距离 (0<=j<nb)
* 如果到达不了则为 -1
* 接下来动态规划
* dp[mask][j]：走到当前状态的最少步数。当前状态为：在第 j 个机关处，机关的触发状态为 mask
* mask 是一个二进制数，因为机关最多 16 个，所以 mask 的长度最长为 16
* 从 mask 的低位开始，每一位数字代表一个机关的触发状态
* 0 代表未触发，1 代表已经触发
* 假设一共 4 个机关，一开始都未触发
* mask 为 0000
* 只触发机关1，mask = 0001
* 只触发机关2，mask = 0010
* 只触发机关4，mask = 1000
* 触发机关1和2，mask = 0011
* 触发机关2和4，mask = 1010
* 那么，总结：
* 只触发 机关i
* mask = 1 << i
* 判断 机关i 是否被触发
* mask & 1 << i != 0
* 假设 i=1，则 1 << i = 0001
* mask 和它与运算，前三位都是 0
* 最后一位，
* 如果已经触发，mask 是 1，则与运行结果不为 0
* 表达式的值为 true
* 如果没有触发，mask 是 0，则与运行结果为 0
* 表达式为 false
* 当前 mask 中，机关i 没有触发，把它变为触发
* mask | 1 << i
* 所有机关都触发的 mask
* finalMask = 1111 = 1 << 4 - 1
* 所以，mask 最小为0，最大为 finalMask = (1 << nb) - 1
* dp 数组的行数为 finalMask + 1 = 1 << nb
* 递归公式：
* dp[mask2][j] = min( dp[mask1][i] + dist[i][j] )
* mask2 是在 mask1 的基础上，只触发了 j
* 那显然，mask2 是由 mask1 决定的
* 但 mask1 和 i 的关系是多对多，谁都不决定谁
* j 在 min 中是固定的
* 所以这个 min，要对 mask1 和 i 遍历
* 也就是填写 dp 的最外面的两层 for 循环
* 最终，dp得到 dp[finalMask][i]
* 即，全部机关都被触发，且当前站在 机关i 处
* 到目前为止走得最短步数
* 边界情况：
* dist[i][nb] 是从 起点 到 机关i 的最短距离
* 那不就是走到 只触发 机关i 且当前在 机关i 的最短距离
* dp[1 << i][i] = dist[i][nb]
* 最终答案
* min( dp[finalMask][i] + dist[i][终点])
* 遍历 i 即可！
* 无法达成的情形：
* 因为从起点到终点需要经过所有的机关
* 所以只要有一个机关，是从起点（经历石头）或者终点不可达的，
* 则任务无法完成，返回 -1

本人代码：

|  |
| --- |
| class Solution {  public:      int n,m;      vector<vector<int>> bfs(int x, int y, vector<string>& maze) {          int coor[4][2] = {0,1,0,-1,-1,0,1,0};//上下左右          vector<vector<int>> arr(n, vector<int>(m, -1));          arr[x][y] = 0;          queue<pair<int,int>> q;          q.push({x,y});          while(!q.empty()) {              int xx = q.front().first;              int yy = q.front().second;              q.pop();              for (int i = 0; i < 4; i++) {                  int row = xx + coor[i][0];                  int col = yy + coor[i][1];                  if (row >= n || col >= m || row < 0 || col < 0)                      continue;                  if (arr[row][col] == -1 && maze[row][col] != '#') {                      arr[row][col] = arr[xx][yy] + 1;                      q.push({row,col});                  }              }          }          return arr;      }      int minimalSteps(vector<string>& maze) {          n = maze.size();          m = maze[0].size();          // 存储机关 & 石头的坐标          vector<pair<int, int>> buttons, stones;          // 记录起点 & 终点的坐标          int sx, sy, tx, ty;          for (int i = 0; i < n; i++) {              for (int j = 0; j < m; j++) {                  if (maze[i][j] == 'M') {                      buttons.push\_back({i, j});                  }                  if (maze[i][j] == 'O') {                      stones.push\_back({i, j});                  }                  if (maze[i][j] == 'S') {                      sx = i;                      sy = j;                  }                  if (maze[i][j] == 'T') {                      tx = i;                      ty = j;                  }              }          }          int nb = buttons.size();          int ns = stones.size();          // 记录起点到其他点的距离          vector<vector<int>> start\_dist = bfs(sx, sy, maze);          // 边界情况：没有机关          if (nb == 0) {              return start\_dist[tx][ty];          }          // 从某个机关到其他机关 / 起点与终点的最短距离          // dist[i][nb] 表示从 机关i 到 起点 的最短距离          // dist[i][nb+1] 表示从 机关i 到 终点 的最短距离          vector<vector<int>> dist(nb, vector<int>(nb + 2, -1));          // 中间结果          vector<vector<vector<int>>> dd(nb);          for (int i = 0; i < nb; i++) {              // 计算i号按钮到所有点的最短距离              vector<vector<int>> d = bfs(buttons[i].first, buttons[i].second, maze);              dd[i] = d;              // 从某个点到终点不需要拿石头              dist[i][nb + 1] = d[tx][ty];// i号按钮到终点的距离          }          for (int i = 0; i < nb; i++) {              int tmp = -1;              // 找出起点到某一处石堆再到i号按钮的最短距离              for (int k = 0; k < ns; k++) {                  int mid\_x = stones[k].first, mid\_y = stones[k].second;                  // i号按钮可以走到k号石头并且起点可以到k号石头                  if (dd[i][mid\_x][mid\_y] != -1 && start\_dist[mid\_x][mid\_y] != -1) {                      if (tmp == -1 || tmp > dd[i][mid\_x][mid\_y] + start\_dist[mid\_x][mid\_y]) {                          tmp = dd[i][mid\_x][mid\_y] + start\_dist[mid\_x][mid\_y];                      }                  }              }              //i号机关到起点的距离（途径某一处石堆）              dist[i][nb] = tmp;              // 找出i号按钮到某处石堆再到j号按钮的最短距离              for (int j = i + 1; j < nb; j++) {                  int mn = -1;                  for (int k = 0; k < ns; k++) {                      int mid\_x = stones[k].first, mid\_y = stones[k].second;                      // i号按钮可以走到k号石头并且j号按钮可以走到k号石头                      if (dd[i][mid\_x][mid\_y] != -1 && dd[j][mid\_x][mid\_y] != -1) {                          if (mn == -1 || mn > dd[i][mid\_x][mid\_y] + dd[j][mid\_x][mid\_y]) {                              mn = dd[i][mid\_x][mid\_y] + dd[j][mid\_x][mid\_y];                          }                      }                  }                  dist[i][j] = mn;                  dist[j][i] = mn;              }          }          // 无法达成的情形(题目要求起点到终点需要经过所有机关)          // 只要有一个机关从起点不可达，或者无法到达终点，就返回-1          for (int i = 0; i < nb; i++) {              if (dist[i][nb] == -1 || dist[i][nb + 1] == -1)                  return -1;          }          // dp 数组， -1 代表没有遍历到          // dp[mask][j]：走到当前状态的最少步数。当前状态为：在第 j 个机关处，机关的触发状态为 mask          vector<vector<int>> dp(1 << nb, vector<int>(nb, -1));          for (int i = 0; i < nb; i++) {              dp[1 << i][i] = dist[i][nb];          }          //mask是二进制数，每一位代表一个按钮的触发状态          // 由于更新的状态都比未更新的大，所以直接从小到大遍历即可          for (int mask = 1; mask < (1 << nb); mask++) {              for (int i = 0; i < nb; i++) {                  // 当前 dp 是合法的                  if (mask & (1 << i)) {                      for (int j = 0; j < nb; j++) {                          // j 不在 mask 里                          if (!(mask & (1 << j))) {                              //将j号按钮变为触发状态                              int next = mask | (1 << j);                              if (dp[next][j] == -1 || dp[next][j] > dp[mask][i] + dist[i][j]) {                                  dp[next][j] = dp[mask][i] + dist[i][j];                              }                          }                      }                  }              }          }          int res = -1;          // 机关全部触发          int final\_mask = (1 << nb) - 1;          // 找到一条从某机关到出口的最短距离(机关已经全部触发并算出了全部触发的最短步数)          for (int i = 0; i < nb; i++) {              if (res == -1 || res > dp[final\_mask][i] + dist[i][nb + 1]) {                  res = dp[final\_mask][i] + dist[i][nb + 1];              }          }          return res;      }  }; |

## [312. 戳气球](https://leetcode-cn.com/problems/burst-balloons/)

### 1 原题目

有 n 个气球，编号为0 到 n-1，每个气球上都标有一个数字，这些数字存在数组 nums 中。

现在要求你戳破所有的气球。如果你戳破气球 i ，就可以获得 nums[left] \* nums[i] \* nums[right] 个硬币。 这里的 left 和 right 代表和 i 相邻的两个气球的序号。注意当你戳破了气球 i 后，气球 left 和气球 right 就变成了相邻的气球。

求所能获得硬币的最大数量。

**说明:**

你可以假设 nums[-1] = nums[n] = 1，但注意它们不是真实存在的所以并不能被戳破。

0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100

**示例:**

输入: [3,1,5,8]

输出: 167

解释: nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []

  coins = 3\*1\*5 + 3\*5\*8 + 1\*3\*8 + 1\*8\*1 = 167

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题目动态规划解法其实有点像自底向上，题目要求是戳气球，而我们是假设一开始只有两个边界的气球，每次往里面加气球，在加起球的过程中寻找并计算最大值。

参考链接：<https://leetcode-cn.com/problems/burst-balloons/solution/zhe-ge-cai-pu-zi-ji-zai-jia-ye-neng-zuo-guan-jian-/>

本人代码：

|  |
| --- |
| class Solution {  public:      int maxCoins(vector<int>& nums) {          int n = nums.size();          vector<vector<int>> dp(n + 2,vector<int>(n + 2,0));          //创建一个辅助数组，并在首尾各添加1，方便处理边界情况          vector<int> temp(n + 2);          temp[0] = 1;          temp[n + 1] = 1;          for(int i = 1; i <= n; i++)              temp[i] = nums[i - 1];           //len表示开区间长度（对每一个区间长度进行循环）          for(int len = 3; len <= n + 2; len++) {              //l表示开区间左端点,l + len - 1则表示开区间右端点              for(int l = 0; l <= n + 2 - len; l++) {                  //k为开区间内的索引（代表区间最后一个被戳破的气球）                  for(int k = l + 1; k < l + len - 1; k++) {                      int left = dp[l][k];                      int right = dp[k][l + len - 1];                      int sum = left + right + temp[k] \* temp[l] \* temp[l + len - 1];                      dp[l][l + len - 1] = max(dp[l][l + len - 1],sum);//取金币最大值                  }              }          }          return dp[0][n + 1];      }  }; |

## [174. 地下城游戏](https://leetcode-cn.com/problems/dungeon-game/)

### 1 原题目

一些恶魔抓住了公主（P）并将她关在了地下城的右下角。地下城是由 M x N 个房间组成的二维网格。我们英勇的骑士（K）最初被安置在左上角的房间里，他必须穿过地下城并通过对抗恶魔来拯救公主。

骑士的初始健康点数为一个正整数。如果他的健康点数在某一时刻降至 0 或以下，他会立即死亡。

有些房间由恶魔守卫，因此骑士在进入这些房间时会失去健康点数（若房间里的值为负整数，则表示骑士将损失健康点数）；其他房间要么是空的（房间里的值为 0），要么包含增加骑士健康点数的魔法球（若房间里的值为正整数，则表示骑士将增加健康点数）。

为了尽快到达公主，骑士决定每次只向右或向下移动一步。

**编写一个函数来计算确保骑士能够拯救到公主所需的最低初始健康点数。**

例如，考虑到如下布局的地下城，如果骑士遵循最佳路径 右 -> 右 -> 下 -> 下，则骑士的初始健康点数至少为 7。

|  |  |  |
| --- | --- | --- |
| -2 (K) | -3 | 3 |
| -5 | -10 | 1 |
| 10 | 30 | -5 (P) |

**说明:**

骑士的健康点数没有上限。

任何房间都可能对骑士的健康点数造成威胁，也可能增加骑士的健康点数，包括骑士进入的左上角房间以及公主被监禁的右下角房间。

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

看到本题你可能会想到直接左上到右下进行动态规划，但实际上，这样算出来的结果不一定是对的，你这样确实能保证和最小，但是路径过程中的健康值你没法使它保持在大于0的状态，就比如说你这样求和动态规划右下角结果是-5，你知道健康值最低是6，但可能第一步减10健康值，所以这就是这个的缺点。

所以，我们可以利用从右下到左上来进行动态规划，这样就保证了，在动态规划的过程中，路途中不会出现健康值小于等于0并且顺利求出最少所需健康值，所以此题动态规划的思路主要就是

假如你下一步走到的是行为n，列为m的盒子中，分两种情况

当前盒子值小于0，所以你走进来之前你的健康值要比这个负数多一（不会死去）

当前盒子大于等于0，所以你走进来之前你的生命值要大于等于1（要不然死了就走不过来）

并且题意告诉我们只能向右向下走，又因为我们找的是最小生命初值，所以我们要在下一步中（右，下）找寻最小的那个，并且生命值要大于等于0，所以动态方程就是

dp[i][j]=max(min(dp[i+1][j],dp[i][j+1])−dungeon(i,j),1)

本人代码：

|  |
| --- |
| class Solution {  public:      int calculateMinimumHP(vector<vector<int>>& dungeon) {          int n = dungeon.size(),m = dungeon[0].size();          vector<vector<int>> dp(n,vector<int>(m));          //dp[i][j] 表示从坐标 (i,j) 到终点所需的最小初始值。          //换句话说，当我们到达坐标 (i,j) 时，如果此时我们的路径和不小于dp[i][j]，我们就能到达终点          dp[n - 1][m - 1] =  max(1 - dungeon[n - 1][m - 1],1);          for(int j = m - 2; j >= 0; --j) {//最后一行先初始化              dp[n - 1][j] = max(dp[n - 1][j + 1] - dungeon[n - 1][j],1);          }          for(int i = n - 2; i >= 0; --i) {//最后一列先初始化              dp[i][m - 1] = max(dp[i + 1][m - 1] - dungeon[i][m - 1],1);          }          //从右下到左上进行动态规划          for(int i = n - 2; i >= 0; --i) {              for(int j = m - 2; j >= 0; --j) {                  int minn = min(dp[i][j + 1],dp[i + 1][j]);                  dp[i][j] = max(minn - dungeon[i][j],1);              }          }          return dp[0][0];      }  }; |

## [44. 通配符匹配](https://leetcode-cn.com/problems/wildcard-matching/)

### 1 原题目

给定一个字符串 (s) 和一个字符模式 (p) ，实现一个支持 '?' 和 '\*' 的通配符匹配。

'?' 可以匹配任何单个字符。

'\*' 可以匹配任意字符串（包括空字符串）。

两个字符串完全匹配才算匹配成功。

**说明:**

s 可能为空，且只包含从 a-z 的小写字母。

p 可能为空，且只包含从 a-z 的小写字母，以及字符 ? 和 \*。

**示例 1:**

输入:

s = "aa"

p = "a"

输出: false

解释: "a" 无法匹配 "aa" 整个字符串。

**示例 2:**

输入:

s = "aa"

p = "\*"

输出: true

解释: '\*' 可以匹配任意字符串。

**示例 3:**

输入:

s = "cb"

p = "?a"

输出: false

解释: '?' 可以匹配 'c', 但第二个 'a' 无法匹配 'b'。

**示例 4:**

输入:

s = "adceb"

p = "\*a\*b"

输出: true

解释: 第一个 '\*' 可以匹配空字符串, 第二个 '\*' 可以匹配字符串 "dce".

**示例 5:**

输入:

s = "acdcb"

p = "a\*c?b"

输出: false

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://leetcode-cn.com/problems/wildcard-matching/solution/tong-pei-fu-pi-pei-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      bool isMatch(string s, string p) {          int m = s.size();          int n = p.size();          //dp[i][j]表示字符串s的前i个字符和模式p的前j个字符是否能匹配          vector<vector<int>> dp(m + 1, vector<int>(n + 1));          //定义空字符串可以匹配          dp[0][0] = 1;          //因为'\*'可匹配空字符串，所以开头连续的'\*'是可以匹配s前0个字符          for (int i = 1; i <= n; i++) {              if (p[i - 1] == '\*') {                  dp[0][i] = 1;              }              else {                  break;              }          }          for (int i = 1; i <= m; i++) {              for (int j = 1; j <= n; j++) {                  if (p[j - 1] == '\*') {                      dp[i][j] = dp[i][j - 1] | dp[i - 1][j];//或运算（'\*'匹配空和任意字符串两种情况）                  }                  else if (p[j - 1] == '?' || s[i - 1] == p[j - 1]) {                      dp[i][j] = dp[i - 1][j - 1];                  }              }          }          return dp[m][n];      }  }; |

## [面试题46. 把数字翻译成字符串](https://leetcode-cn.com/problems/ba-shu-zi-fan-yi-cheng-zi-fu-chuan-lcof/)

### 1 原题目

给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a” ，1 翻译成 “b”，……，11 翻译成 “l”，……，25 翻译成 “z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。

**示例 1:**

输入: 12258

输出: 5

解释: 12258有5种不同的翻译，分别是"bccfi", "bwfi", "bczi", "mcfi"和"mzi"

**提示：**

0 <= num < 2^31

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

本人当时解出来只是找规律找出来的动态方程，具体推导看链接吧，我讲不清。

参考链接：<https://leetcode-cn.com/problems/ba-shu-zi-fan-yi-cheng-zi-fu-chuan-lcof/solution/ba-shu-zi-fan-yi-cheng-zi-fu-chuan-by-leetcode-sol/>

本人代码：

|  |
| --- |
| class Solution {  public:      int translateNum(int num) {          if(num == 0)              return 1;          int numm[12],dp[12];//numm存储每一位的数字，dp[i]表示数字第i位（包含i）之前翻译数          int bit = log10(num) + 1;//计算num的位数          int i = bit - 1;          while(num) {              numm[i--] = num % 10;              num /= 10;          }          dp[0] = 1;//一位数肯定是可以翻译成一种          //之后便进行动态规划，当前数字和前一位数字组成的数字是否大于25          //大于25，dp[i] = dp[i-1]，小于等于25，dp[i] = dp[i-1]+dp[i-2];          if(bit >= 2 && numm[0] \* 10 + numm[1] <= 25)              dp[1] = 2;          else              dp[1] = 1;          for(i = 2; i <= bit - 1; i++) {              if(numm[i - 1] != 0 && numm[i - 1] \* 10 + numm[i] <= 25)                  dp[i] = dp[i - 1] + dp[i - 2];              else                  dp[i] = dp[i - 1];          }          return dp[bit - 1];      }  }; |

## [837. 新21点](https://leetcode-cn.com/problems/new-21-game/)

### 1 原题目

爱丽丝参与一个大致基于纸牌游戏 “21点” 规则的游戏，描述如下：

爱丽丝以 0 分开始，并在她的得分少于 K 分时抽取数字。 抽取时，她从 [1, W] 的范围中随机获得一个整数作为分数进行累计，其中 W 是整数。 每次抽取都是独立的，其结果具有相同的概率。

当爱丽丝获得不少于 K 分时，她就停止抽取数字。 爱丽丝的分数不超过 N 的概率是多少？

**示例 1：**

输入：N = 10, K = 1, W = 10

输出：1.00000

说明：爱丽丝得到一张卡，然后停止。

**示例 2：**

输入：N = 6, K = 1, W = 10

输出：0.60000

说明：爱丽丝得到一张卡，然后停止。

在 W = 10 的 6 种可能下，她的得分不超过 N = 6 分。

**示例 3：**

输入：N = 21, K = 17, W = 10

输出：0.73278

**提示：**

0 <= K <= N <= 10000

1 <= W <= 10000

如果答案与正确答案的误差不超过 10^-5，则该答案将被视为正确答案通过。

此问题的判断限制时间已经减少。

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

dp[x]是手上牌的分数为x时获胜的概率（分数大于等于K停止抽牌，分数小于等于N为获胜）既然如此，我们手里最开始的分数一定为0，而且在1-W中抽数字概率相同，所以说，我们可以知道，分数0获胜的概率是1-w（实际上是0加1-W）的概率和除W，所以可以推导出状态方程。然后我们知道，当手里牌为大于等于K，小于等于N时，获胜概率都为1（否则为0）。于是我们可以利用动态规划方程自顶向下求解。

参考链接：<https://leetcode-cn.com/problems/new-21-game/solution/huan-you-bi-zhe-geng-jian-dan-de-ti-jie-ma-tian-ge/>

本人代码：

|  |
| --- |
| class Solution {  public:      double new21Game(int N, int K, int W) {          double sum = 0,i;          vector<double> dp(K + W,0);          //dp[x]为她手上牌面为x时，能获胜的概率是：dp[x]=1/w \* (dp[x+1]+dp[x+2]+dp[x+3]...+dp[x+w])          //获胜条件是小于等于N，而分数大于等于K就停止抽牌          //所以手里牌为大于等于K，小于等于N时，获胜概率都为1          for (i = K; i < K + W && i <= N; i++) {              dp[i] = 1.0;              sum += dp[i];          }          //从式子可以看出，我们可以从后往前动态规划          for(i = K - 1; i >= 0; i--) {              dp[i] = sum / W;//求值              sum = sum - dp[i + W] + dp[i];//减去末尾的，加上现在的（维持是W个之和）          }          return dp[0];      }  }; |

## [221. 最大正方形](https://leetcode-cn.com/problems/maximal-square/)

### 1 原题目

在一个由 0 和 1 组成的二维矩阵内，找到只包含 1 的最大正方形，并返回其面积。

**示例:**

输入:

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

输出: 4

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用动态规划，dp[i][j]记录的是以i，j为右下角的正方形的边长。

dp[i][j]如果为0那么dp[i][j]肯定为0

dp[i][j]如果在第0行和第0列那么dp[i][j] = matrix[i][j]

依次每一行的遍历matrix数组，如果matrix[i][j]为1，那当前位置正方形的边长取决于左上角，左，上的最小值加1。

本人代码：

|  |
| --- |
| class Solution {  public:      int maximalSquare(vector<vector<char>>& matrix) {          if(matrix.size() == 0)              return 0;          int row = matrix.size(),col = matrix[0].size();          vector<vector<int>> dp(row, vector<int>(col));          int i,j,maxx = 0;          for(i = 0; i < row; i++) {              for(j = 0; j < col; j++) {                  if(matrix[i][j] == '1') {                      if(i == 0 || j == 0)                          dp[i][j] = 1;                      else                          dp[i][j] = min(min(dp[i - 1][j - 1],dp[i][j - 1]),dp[i - 1][j]) + 1;//左上角，左，上                  }                  maxx = max(dp[i][j],maxx);              }          }          return maxx \* maxx;      }  }; |

## [53. 最大子序和](https://leetcode-cn.com/problems/maximum-subarray/)

### 1 原题目

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

**示例:**

输入: [-2,1,-3,4,-1,2,1,-5,4],

输出: 6

解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

**进阶:**

如果你已经实现复杂度为 O(n) 的解法，尝试使用更为精妙的分治法求解。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

动态规划方程dp[i] = max(dp[i-1] + nums[i],nums[i])，dp[i]代表以第i个数结尾的连续子数组的最大和,最终结果就是在dp数组中选择最大值。利用动态方程可以写出空间复杂度为O(n)的写法，但是也可以写成空间复杂度为O(1)的写法。

说一下思路：

假定有数组[3,6,-1,2,5,6,9,3,2]

以第0个数结尾的最大连续子数组肯定是自己本身

以第1个数结尾的最大连续子数组肯定是第一个数或第一个数和第零个数相加这两者之一

以第2个数结尾的最大子数组有三种：第二个数自己，第二个数加第一个数，第二个数加第一个数加第零个数。

简单观察下上面的推理，似乎可以发现：

以第2个数结尾的最大子数组有三种，后两种似乎可以利用上一项（以第1个数结尾的最大连续子数组）的结果，所以推出动态规划方程并可以改写成空间复杂度O(1)的写法(因为动态规划方程只利用了前一项求当前项，所以用个常量记录前一项就省去了开dp数组)

本人代码：

|  |
| --- |
| class Solution {  public:      int maxSubArray(vector<int>& nums) {          int prev = 0,maxx = nums[0];          //动态规划方程dp[i] = max(dp[i-1] + nums[i],nums[i])          //dp[i]代表以第i个数结尾的连续子数组的最大和,最终结果就是在dp数组中选择最大值          //观察方程发现可以变成空间消耗为1的写法，如下：          for(int i = 0; i < nums.size(); i++) {              prev = max(prev + nums[i],nums[i]);              maxx = max(prev,maxx);          }          return maxx;      }  }; |

## [72. 编辑距离](https://leetcode-cn.com/problems/edit-distance/)

### 1 原题目

给定两个单词 word1 和 word2，计算出将 word1 转换成 word2 所使用的最少操作数 。

你可以对一个单词进行如下三种操作：

插入一个字符

删除一个字符

替换一个字符

**示例 1:**

输入: word1 = "horse", word2 = "ros"

输出: 3

解释:

horse -> rorse (将 'h' 替换为 'r')

rorse -> rose (删除 'r')

rose -> ros (删除 'e')

**示例 2:**

输入: word1 = "intention", word2 = "execution"

输出: 5

解释:

intention -> inention (删除 't')

inention -> enention (将 'i' 替换为 'e')

enention -> exention (将 'n' 替换为 'x')

exention -> exection (将 'n' 替换为 'c')

exection -> execution (插入 'u')

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

参考链接：<https://mp.weixin.qq.com/s?__biz=Mzg2NzA4MTkxNQ==&mid=2247486294&idx=1&sn=dd8968700d19ea8b1db29065dc2f7b01&chksm=ce404082f937c994444393163a2bf53f32e18f9a752aad610c3d4988f31c328cae203a84809b&scene=21#wechat_redirect>

本人代码：

class Solution {

    public int minDistance(String word1, String word2) {

        int n1 = word1.length();

        int n2 = word2.length();

        //当字符串 word1 的长度为 i，字符串 word2 的长度为 j 时，将 word1 转化为 word2 所使用的最少操作次数为 dp[i] [j]

        int [][] dp = new int[n1 + 1][n2 + 1];

        int i,j;

        //当word1字符串为空时，只能进行插入操作

        for(i = 1; i <= n2; i++)

            dp[0][i] = dp[0][i - 1] + 1;

        //当word2字符串为空时，只能进行删除操作

        for(i = 1; i <= n1; i++)

            dp[i][0] = dp[i - 1][0] + 1;

        //遍历两个字符串

        for(i = 1; i <= n1; i++) {

            for(j = 1; j <= n2; j++) {

                if(word1.charAt(i - 1) == word2.charAt(j - 1))//字符相等，不需要进行任何操作

                    dp[i][j] = dp[i - 1][j - 1];

                else

                    dp[i][j] = Math.min(Math.min(dp[i - 1][j],dp[i][j - 1]),dp[i - 1][j - 1]) + 1;//在删除，插入，替换中选择一个最小的

            }

        }

        return dp[n1][n2];

    }

}

## [300. 最长上升子序列](https://leetcode-cn.com/problems/longest-increasing-subsequence/)

### 1 原题目

给定一个无序的整数数组，找到其中最长上升子序列的长度。

**示例:**

输入: [10,9,2,5,3,7,101,18]

输出: 4

解释: 最长的上升子序列是 [2,3,7,101]，它的长度是 4。

**说明:**

可能会有多种最长上升子序列的组合，你只需要输出对应的长度即可。

你算法的时间复杂度应该为 O(n2) 。

**进阶:** 你能将算法的时间复杂度降低到 O(n log n) 吗?

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

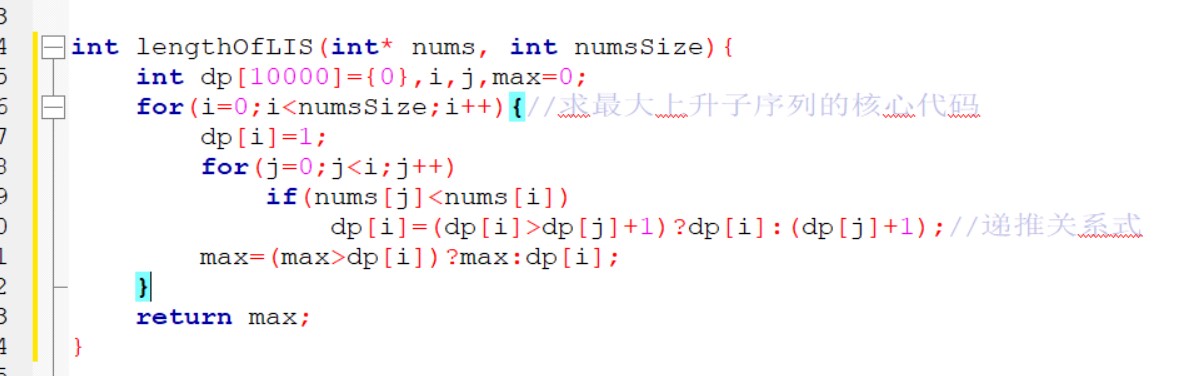
动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

参考这个链接：<https://blog.csdn.net/ltrbless/article/details/81318935>（我就是参考这个的）

本人代码：



## [97. 交错字符串](https://leetcode-cn.com/problems/interleaving-string/)

### 1 原题目

给定三个字符串 s1, s2, s3, 验证 s3 是否是由 s1 和 s2 交错组成的。

**示例 1:**

输入: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"

输出: true

**示例 2:**

输入: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"

输出: false

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://leetcode-cn.com/problems/interleaving-string/solution/jiao-cuo-zi-fu-chuan-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      bool isInterleave(string s1, string s2, string s3) {          if(s1.length() + s2.length() != s3.length())//长度不等时直接返回false              return false;          //dp[i][j]表示s1的前i个字符和s2的前j个字符能否交叉组成s3前i+j个字符          vector<vector<int>> dp(s1.length() + 1,vector<int>(s2.length() + 1,0));          dp[0][0] = 1;//前0个字符肯定是可以匹配的          for(int i = 0; i <= s1.length(); i++) {              for(int j = 0; j <= s2.length(); j++) {                  if(i > 0 && s1[i - 1] == s3[i + j - 1] && dp[i - 1][j])                      dp[i][j] = dp[i - 1][j];                  if(j > 0 && s2[j - 1] == s3[i + j - 1] && dp[i][j - 1])                      dp[i][j] = dp[i][j - 1];              }          }          return dp[s1.length()][s2.length()];      }  }; |

## [377. 组合总和 Ⅳ](https://leetcode-cn.com/problems/combination-sum-iv/)

### 1 原题目

给定一个由正整数组成且不存在重复数字的数组，找出和为给定目标正整数的组合的个数。

**示例:**

nums = [1, 2, 3]

target = 4

所有可能的组合为：

(1, 1, 1, 1)

(1, 1, 2)

(1, 2, 1)

(1, 3)

(2, 1, 1)

(2, 2)

(3, 1)

请注意，顺序不同的序列被视作不同的组合。

因此输出为 7。

**进阶：**

如果给定的数组中含有负数会怎么样？

问题会产生什么变化？

我们需要在题目中添加什么限制来允许负数的出现？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

题目参考链接：<https://leetcode-cn.com/problems/combination-sum-iv/solution/dong-tai-gui-hua-python-dai-ma-by-liweiwei1419/>

本人代码：

class Solution {

    public int combinationSum4(int[] nums, int target) {

        int i,j;

        int[] dp = new int[target+1];//自动将数组里的元素初始化为0

        dp[0]=1;//为了算上数据元素等于target这种情况

        for(i=1;i<=target;i++) {

            for(j=0;j<nums.length;j++) {

                if(i-nums[j]>=0)//就是为了在nums数组中找出和为target的组合

                    dp[i]=dp[i]+dp[i-nums[j]];

            }

        }

        return dp[target];

    }

}

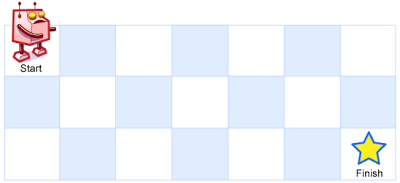
## [62. 不同路径](https://leetcode-cn.com/problems/unique-paths/)

### 1 原题目

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

问总共有多少条不同的路径？



例如，上图是一个7 x 3 的网格。有多少可能的路径？

**说明：**m 和 n 的值均不超过 100。

**示例 1:**

输入: m = 3, n = 2

输出: 3

解释:

从左上角开始，总共有 3 条路径可以到达右下角。

1. 向右 -> 向右 -> 向下

2. 向右 -> 向下 -> 向右

3. 向下 -> 向右 -> 向右

**示例 2:**

输入: m = 7, n = 3

输出: 28

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

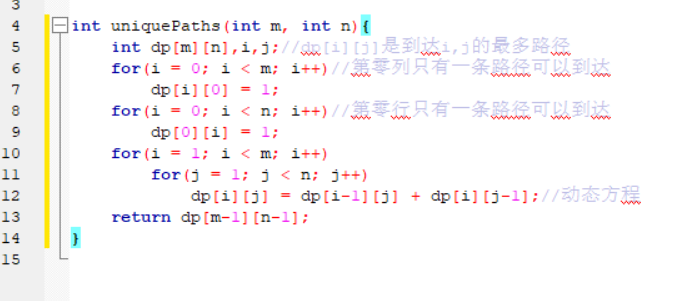
动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

dp[i][j]是到达i,j的最多路径，因为只能往右或者往下走，所以从起点到第零行和第零列的路径只有一条，经过推导可以得到动态方程dp[i][j] = dp[i-1][j] + dp[i][j-1]

本人代码：



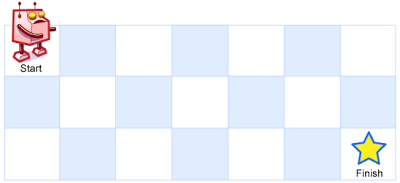
## [63. 不同路径 II](https://leetcode-cn.com/problems/unique-paths-ii/)

### 1 原题目

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？



网格中的障碍物和空位置分别用 1 和 0 来表示。

**说明：**m 和 n 的值均不超过 100。

**示例 1:**

输入:

[

  [0,0,0],

  [0,1,0],

  [0,0,0]

]

输出: 2

解释:

3x3 网格的正中间有一个障碍物。

从左上角到右下角一共有 2 条不同的路径：

1. 向右 -> 向右 -> 向下 -> 向下

2. 向下 -> 向下 -> 向右 -> 向右

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

类似于LeetCode 62题，此题只不过是需要对路径进行一个是否有障碍物的判断，具体请看代码和注释。

本人代码：

class Solution {

    public int uniquePathsWithObstacles(int[][] obstacleGrid) {

        int row = obstacleGrid.length;//题目所给的二维数组的行数

        int col = obstacleGrid[0].length;//题目所给的二维数组的列数

        int[][] dp = new int[row][col];

        int i,j;

        for(i = 0; i < row; i++)//第零列只有一条路径可以到达

            if(obstacleGrid[i][0] == 0)//空位置时

                dp[i][0] = 1;

            else//出现了障碍物那么之后的路径就无法到达

                break;

        for(i = 0; i < col; i++)//第零行只有一条路径可以到达

            if(obstacleGrid[0][i] == 0)//空位置时

                dp[0][i] = 1;

            else//出现了障碍物那么之后的路径就无法到达

                break;

        for(i = 1; i < row; i++)//遍历题目所给的二维数组

            for(j = 1; j < col; j++) {

                if(obstacleGrid[i][j] == 1)//遇到障碍物就跳过

                    continue;

                //因为只能向下走或者向右走，所以到达当前位置的路径只有从上边过来和左边过来两种

                //然后再判断一下这两种是否有障碍物就可以了

                if(obstacleGrid[i - 1][j] == 0 && obstacleGrid[i][j-1] == 0)

                    dp[i][j] = dp[i-1][j] + dp[i][j-1];//动态方程

                if(obstacleGrid[i - 1][j] == 0 && obstacleGrid[i][j-1] != 0)

                    dp[i][j] = dp[i-1][j];

                if(obstacleGrid[i - 1][j] != 0 && obstacleGrid[i][j-1] == 0)

                    dp[i][j] = dp[i][j-1];

            }

        return dp[row-1][col-1];

    }

}

## [64. 最小路径和](https://leetcode-cn.com/problems/minimum-path-sum/)

### 1 原题目

给定一个包含非负整数的 m x n 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

**说明**：每次只能向下或者向右移动一步。

**示例:**

输入:

[

  [1,3,1],

[1,5,1],

[4,2,1]

]

输出: 7

解释: 因为路径 1→3→1→1→1 的总和最小。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

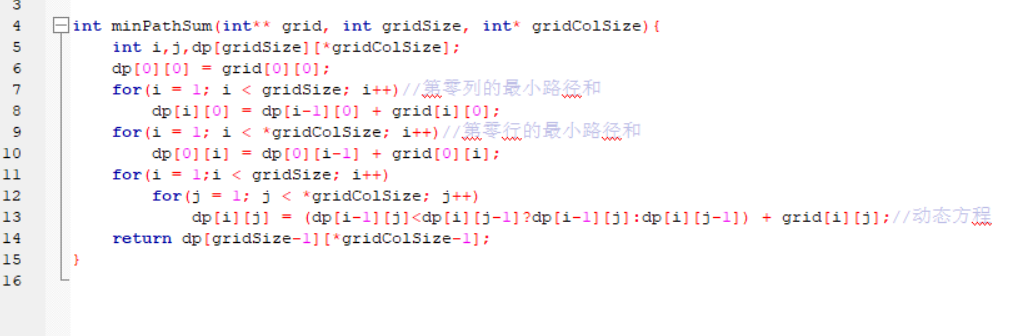
动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

dp[i][j]是存储到达i,j的所有路径中和最少的，因为只能往右或者往下走，所以从起点到第零行和第零列的路径只有一条，所以第零行和第零列的和就一直累加就ok了。经过推导可以得到动态方程dp[i][j] = min ( dp[i-1][j] , dp[i][j-1] ) + grid[i][j]。grid[i][j]就是当前位置数组中的值

本人代码：



## [70. 爬楼梯](https://leetcode-cn.com/problems/climbing-stairs/)

### 1 原题目

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

**注意：**给定 n 是一个正整数。

**示例 1：**

输入： 2

输出： 2

解释： 有两种方法可以爬到楼顶。

1. 1 阶 + 1 阶

2. 2 阶

**示例 2：**

输入： 3

输出： 3

解释： 有三种方法可以爬到楼顶。

1. 1 阶 + 1 阶 + 1 阶

2. 1 阶 + 2 阶

3. 2 阶 + 1 阶

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

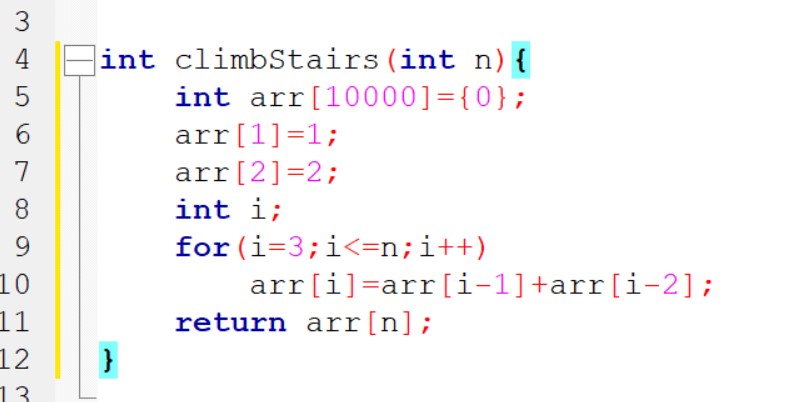
动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

这道题有规律，从第三项开始，每一项是前两项的和。

本人代码：



## [279. 完全平方数](https://leetcode-cn.com/problems/perfect-squares/)

### 1 原题目

给定正整数 n，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n。你需要让组成和的完全平方数的个数最少。

**示例 1:**

输入: n = 12

输出: 3

解释: 12 = 4 + 4 + 4.

**示例 2:**

输入: n = 13

输出: 2

解释: 13 = 4 + 9.

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

参考链接：<https://leetcode-cn.com/problems/perfect-squares/solution/hua-jie-suan-fa-279-wan-quan-ping-fang-shu-by-guan/>

本人代码：

class Solution {

    public int numSquares(int n) {

        int[] dp = new int[n + 1]; // 默认初始化值都为0

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

            dp[i] = i; // 最坏的情况就是每次+1

            for (int j = 1; i - j \* j >= 0; j++) {

                dp[i] = Math.min(dp[i], dp[i - j \* j] + 1); // 动态转移方程

            }

        }

        return dp[n];

    }

}

## [198. 打家劫舍](https://leetcode-cn.com/problems/house-robber/)

### 1 原题目

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你在不触动警报装置的情况下，能够偷窃到的最高金额。

**示例 1:**

输入: [1,2,3,1]

输出: 4

解释: 偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。

  偷窃到的最高金额 = 1 + 3 = 4 。

**示例 2:**

输入: [2,7,9,3,1]

输出: 12

解释: 偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9)，接着偷窃 5 号房屋 (金额 = 1)。

  偷窃到的最高金额 = 2 + 9 + 1 = 12 。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

参考此链接的解释：<https://leetcode-cn.com/problems/house-robber/solution/hua-jie-suan-fa-198-da-jia-jie-she-by-guanpengchn/>

本人代码：

class Solution {

    public int rob(int[] nums) {

        int dp[] = new int[nums.length + 2];//默认初始化为0

        for(int i = 0; i < nums.length; i++)

            dp[i + 2] = Math.max(dp[i] + nums[i], dp[i + 1]);//动态方程

        return dp[nums.length + 1];

    }

}

## [213. 打家劫舍 II](https://leetcode-cn.com/problems/house-robber-ii/)

### 1 原题目

你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都围成一圈，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你在不触动警报装置的情况下，能够偷窃到的最高金额。

**示例 1:**

输入: [2,3,2]

输出: 3

解释: 你不能先偷窃 1 号房屋（金额 = 2），然后偷窃 3 号房屋（金额 = 2）, 因为他们是相邻的。

**示例 2:**

输入: [1,2,3,1]

输出: 4

解释: 你可以先偷窃 1 号房屋（金额 = 1），然后偷窃 3 号房屋（金额 = 3）。

  偷窃到的最高金额 = 1 + 3 = 4 。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

这道题本质上就是198这道题。只不过在此题我们对数组的遍历要分两种情况。一种是加上首元素，一种是不加首元素，最后在这两种情况中选取一个较大的值。

198题参考此链接的解释：<https://leetcode-cn.com/problems/house-robber/solution/hua-jie-suan-fa-198-da-jia-jie-she-by-guanpengchn/>

本人代码：

class Solution {

    public int rob(int[] nums) {

        if(nums.length == 1)//只有一个房屋时

            return nums[0];

        int dp[] = new int[nums.length + 2];//默认初始化为0

        int dp1[] = new int[nums.length + 2];

        for (int i = 1; i < nums.length; i++)//去掉nums的头元素再遍历nums数组

            dp[i + 2] = Math.max(dp[i] + nums[i], dp[i + 1]);//动态方程

        for (int i = 0; i < nums.length - 1; i++)//去掉nums的尾元素遍历nums数组

            dp1[i + 2] = Math.max(dp1[i] + nums[i], dp1[i + 1]);

        return Math.max(dp[nums.length + 1],dp1[nums.length]);//在盗窃第一个房屋和不盗窃第一个房屋这两种情况中选择一个大的

    }

}

## [337. 打家劫舍 III](https://leetcode-cn.com/problems/house-robber-iii/)

### 1 原题目

在上次打劫完一条街道之后和一圈房屋后，小偷又发现了一个新的可行窃的地区。这个地区只有一个入口，我们称之为“根”。 除了“根”之外，每栋房子有且只有一个“父“房子与之相连。一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。 如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。

计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。

**示例 1:**

输入: [3,2,3,null,3,null,1]

3

/ \

2 3

\ \

3 1

输出: 7

解释: 小偷一晚能够盗取的最高金额 = 3 + 3 + 1 = 7.

**示例 2:**

输入: [3,4,5,1,3,null,1]

  3

/ \

4 5

/ \ \

1 3 1

输出: 9

解释: 小偷一晚能够盗取的最高金额 = 4 + 5 = 9.

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

递归（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

我们可以将小偷对于二叉树的偷取分成两种情况，一种是偷取根节点（根节点的左右孩子都无法偷取，但可以偷取根节点左孩子的左右孩子，根节点右孩子的左右孩子），另一种是不偷取根节点（偷根节点的左孩子和右孩子），这两种情况之中选取较大的就是答案，剩下的就是递归了。

本人代码：

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     struct TreeNode \*left;

 \*     struct TreeNode \*right;

 \* };

 \*/

int max(int a,int b) {//比较大小

    if(a > b)

        return a;

    else

        return b;

}

int rob(struct TreeNode\* root){

    if(root == NULL)

            return 0;

    //分别表示左孩子，左孩子的左孩子，左孩子的右孩子，右孩子，右孩子的右孩子，右孩子的左孩子

    int left = 0,leftl = 0,leftr = 0,right = 0,rightr = 0,rightl = 0;

    left = rob(root -> left);//递归计算左孩子的最大值

    right = rob(root -> right);//递归计算右孩子的最大值

    if(root -> left) {//如果左孩子存在

        leftl = rob(root -> left -> left);//递归算出左孩子的左孩子的最大值

        leftr = rob(root -> left -> right);//递归算出左孩子的右孩子的最大值

    }

    if(root -> right) {//如果右孩子存在

        rightl = rob(root -> right -> left);//递归算出右孩子的左孩子的最大值

        rightr = rob(root -> right -> right);//递归算出右孩子的右孩子的最大值

    }

    //在偷取当前根节点和不偷取当前根节点这两种情况中选一个较大的

    return max(left + right,root -> val + leftl + leftr + rightl + rightr);

}

## [309. 最佳买卖股票时机含冷冻期](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/)

### 1 原题目

给定一个整数数组，其中第 i 个元素代表了第 i 天的股票价格 。​

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）:

你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

**示例:**

输入: [1,2,3,0,2]

输出: 3

解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/solution/zui-jia-mai-mai-gu-piao-shi-ji-han-leng-dong-qi-4/>

本人代码：

|  |
| --- |
| class Solution {  public:      int maxProfit(vector<int>& prices) {          if(prices.size() == 0)              return 0;          vector<vector<int>> dp(prices.size(),vector<int>(3,0));          dp[0][0] = -prices[0];          //这里的处于冷冻期指的是在第 ii天结束之后的状态。也就是说：如果第 ii天结束之后处于冷冻期，那么第 i+1 天无法买入股票。          // dp[i][0]: 手上持有股票的最大收益          // dp[i][1]: 手上不持有股票，并且处于冷冻期中的累计最大收益          // dp[i][2]: 手上不持有股票，并且不在冷冻期中的累计最大收益          for(int i = 1; i < prices.size(); i++) {               dp[i][0] = max(dp[i - 1][0],dp[i - 1][2] - prices[i]);               dp[i][1] = dp[i - 1][0] + prices[i];               dp[i][2] = max(dp[i - 1][1],dp[i - 1][2]);          }          return max(dp[prices.size() - 1][1],dp[prices.size() - 1][2]);      }  }; |

## [96. 不同的二叉搜索树](https://leetcode-cn.com/problems/unique-binary-search-trees/)

### 1 原题目

给定一个整数 n，求以 1 ... n 为节点组成的二叉搜索树有多少种？

**示例:**

输入: 3

输出: 5

解释:

给定 n = 3, 一共有 5 种不同结构的二叉搜索树:

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

参考下面两个链接：

<https://leetcode-cn.com/problems/unique-binary-search-trees/solution/bu-tong-de-er-cha-sou-suo-shu-by-leetcode/>

<https://leetcode-cn.com/problems/unique-binary-search-trees/solution/hua-jie-suan-fa-96-bu-tong-de-er-cha-sou-suo-shu-b/>

本人代码：

class Solution {

    public int numTrees(int n) {

        int[] dp = new int[n + 1];

        dp[0] = 1;

        dp[1] = 1;

        for(int i = 2; i <= n; i++)

            for(int j = 1; j <= i; j++)

                dp[i] = dp[i] + dp[j - 1] \* dp[i - j];//动态方程

        return dp[n];

    }

}

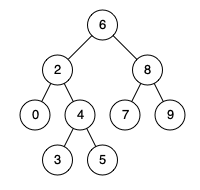
## [235. 二叉搜索树的最近公共祖先](https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-search-tree/)

### 1 原题目

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

例如，给定如下二叉搜索树:  root = [6,2,8,0,4,7,9,null,null,3,5]



**示例 1:**

输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

输出: 6

解释: 节点 2 和节点 8 的最近公共祖先是 6。

**示例 2:**

输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

输出: 2

解释: 节点 2 和节点 4 的最近公共祖先是 2, 因为根据定义最近公共祖先节点可以为节点本身。

**说明:**

所有节点的值都是唯一的。

p、q 为不同节点且均存在于给定的二叉搜索树中。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（java已过）

将两个节点的值与当前的根节点的值进行判断，根据二叉搜索树的性质判断出两个节点相对于当前的根节点所处的位置，然后递归下去就可以找到最近的公共祖先。

具体解释可以看这个链接：<https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-search-tree/solution/er-cha-sou-suo-shu-de-zui-jin-gong-gong-zu-xian--2/>

本人代码：

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

        int parentval = root.val, pval = p.val, qval = q.val;

        if(pval > parentval && qval > parentval)

            return lowestCommonAncestor(root.right,p,q);

        else if(pval < parentval && qval < parentval)

            return lowestCommonAncestor(root.left,p,q);

        else

            return root;

    }

}

## [65. 有效数字](https://leetcode-cn.com/problems/valid-number/)

### 1 原题目

验证给定的字符串是否可以解释为十进制数字。

例如:

"0" => true

" 0.1 " => true

"abc" => false

"1 a" => false

"2e10" => true

" -90e3   " => true

" 1e" => false

"e3" => false

" 6e-1" => true

" 99e2.5 " => false

"53.5e93" => true

" --6 " => false

"-+3" => false

"95a54e53" => false

**说明:**我们有意将问题陈述地比较模糊。在实现代码之前，你应当事先思考所有可能的情况。这里给出一份可能存在于有效十进制数字中的字符列表：

数字 0-9

指数 - "e"

正/负号 - "+"/"-"

小数点 - "."

当然，在输入中，这些字符的上下文也很重要。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

这道题之所以可以通过，全靠提交数量来找自己代码的不足，感谢leetcode的平台可以让我看见错在哪个例子。

声明一下需要注意的：

1. 空字符串
2. 字符串全由空格组成
3. 字符串中出现了除小写字母e以外的其他字母
4. 小写字母e出现了两次
5. 删掉字符串首尾的空格后，字符串中存在有空格
6. 字母e的后面不能有小数点

还有一些我不能用语言很好的描述出来，需要意会，详情请看我的代码。

本人代码：

bool isNumber(char \* s){

//point用来记录小数点是否出现,sign\_e用来记录e是否出现

//num用来记录e之前出现数字的个数，num1用来记录e之后出现数字的个数

int len=strlen(s),i=0,point=0,sign\_e=0,num=0,num1=0,j=len-1;

if(len==0)//字符串长度为空时为错误

return false;

while(i<len&&s[i]==' ')//去掉字符串前面的空格

i++;

if(i==len)//如果字符串全是由空格组成也是错误的

return false;

while(j>=0&&s[j]==' ')//去掉字符串后面的空格

j--;

if(i>j)

return false;

if(s[i]=='-'||s[i]=='+'){//如果去掉空格后，有符号

i=i+1;//i要加1

}

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

if(s[i]==' ')//字符串中出现空格是错误的

return false;

if((s[i]>='a'&&s[i]<='z'&&s[i]!='e')||(s[i]>='A'&&s[i]<='Z'))//出现了除e以外的字母是错误的

return false;

if(s[i]=='-'||s[i]=='+')//字符串中最多可以出现两次这种符号

return false;//一次在去空格后字符串的头部，另一次是紧跟着出现在e的后面，除了这两种情况再出现就是错误的

if((s[i]=='.'&&point==1)||(s[i]=='.'&&sign\_e==1)||(s[i]=='.'&&num==0&&(s[i+1]<'0'||s[i+1]>'9')))

return false;//小数点的出现情况如果是以上几种，就是错误的

if((s[i]>='0'&&s[i]<='9')&&sign\_e==0)

num++;

if((s[i]>='0'&&s[i]<='9')&&sign\_e==1)

num1++;

if(s[i]=='e'){

if(num==0)//e之前如果没数字是错误的

return false;

if(sign\_e==1)//e多次出现是错误的

return false;

sign\_e=1;

if(s[i+1]=='-'||s[i+1]=='+'){

i=i+1;

}

}

if(s[i]=='.'&&point==0)

point=1;

}

if(sign\_e==1&&num1==0)//e之后没数字是错误的

return false;

return true;

}

## [36. 有效的数独](https://leetcode-cn.com/problems/valid-sudoku/)

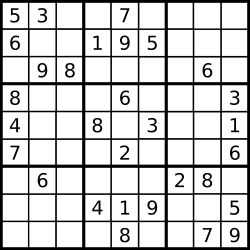
### 1 原题目

判断一个 9x9 的数独是否有效。只需要根据以下规则，验证已经填入的数字是否有效即可。

数字 1-9 在每一行只能出现一次。

数字 1-9 在每一列只能出现一次。

数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。



上图是一个部分填充的有效的数独。

数独部分空格内已填入了数字，空白格用 '.' 表示。

**示例 1:**

输入:

[

["5","3",".",".","7",".",".",".","."],

["6",".",".","1","9","5",".",".","."],

[".","9","8",".",".",".",".","6","."],

["8",".",".",".","6",".",".",".","3"],

["4",".",".","8",".","3",".",".","1"],

["7",".",".",".","2",".",".",".","6"],

[".","6",".",".",".",".","2","8","."],

[".",".",".","4","1","9",".",".","5"],

[".",".",".",".","8",".",".","7","9"]

]

输出: true

**示例 2:**

输入:

[

  ["8","3",".",".","7",".",".",".","."],

  ["6",".",".","1","9","5",".",".","."],

  [".","9","8",".",".",".",".","6","."],

  ["8",".",".",".","6",".",".",".","3"],

  ["4",".",".","8",".","3",".",".","1"],

  ["7",".",".",".","2",".",".",".","6"],

  [".","6",".",".",".",".","2","8","."],

  [".",".",".","4","1","9",".",".","5"],

  [".",".",".",".","8",".",".","7","9"]

]

输出: false

解释: 除了第一行的第一个数字从 5 改为 8 以外，空格内其他数字均与 示例1 相同。

但由于位于左上角的 3x3 宫内有两个 8 存在, 因此这个数独是无效的。

**说明:**

一个有效的数独（部分已被填充）不一定是可解的。

只需要根据以上规则，验证已经填入的数字是否有效即可。

给定数独序列只包含数字 1-9 和字符 '.' 。

给定数独永远是 9x9 形式的。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

这种做法比较暴力，就是每行，每列，每个小正方形都去判断是不是满足数字出现一次这个条件。

本人代码：

bool isValidSudoku(char\*\* board, int boardSize, int\* boardColSize){

int i,j,k;

for(i=0;i<boardSize;i++){//判断每一行是否满足条件

int arr[10]={0};//用来记录数字出现的次数

for(j=0;j<\* boardColSize;j++){

if(board[i][j]>='0'&&board[i][j]<='9')

arr[board[i][j]-'0']++;

}

for(k=0;k<10;k++)

if(arr[k]>=2)//某一个数字出现了两次以上就不是一个正确的数独

return false;

}

for(i=0;i<\* boardColSize;i++){//判断每一列是否满足条件

int arr[10]={0};

for(j=0;j<boardSize;j++){

if(board[j][i]>='0'&&board[j][i]<='9')

arr[board[j][i]-'0']++;

}

for(k=0;k<10;k++)

if(arr[k]>=2)

return false;

}

int arr[10]={0};

for(i=0;i<boardSize;i++){//判断左边的三个小正方形中的数字是否满足条件

for(j=0;j<3;j++){

if(board[i][j]>='0'&&board[i][j]<='9')

arr[board[i][j]-'0']++;

}

if((i+1)%3==0){//每个正方行三行三列，这个if用来判断是否已经遍历了一个正方形

for(k=0;k<10;k++)

if(arr[k]>=2)

return false;

memset(arr,0,sizeof(arr));//数组置为0的函数

}

}

for(i=0;i<boardSize;i++){//判断中间的三个小正方形中的数字是否满足条件

for(j=3;j<6;j++){

if(board[i][j]>='0'&&board[i][j]<='9')

arr[board[i][j]-'0']++;

}

if((i+1)%3==0){

for(k=0;k<10;k++)

if(arr[k]>=2)

return false;

memset(arr,0,sizeof(arr));

}

}

for(i=0;i<boardSize;i++){//判断右边的三个小正方形中的数字是否满足条件

for(j=6;j<9;j++){

if(board[i][j]>='0'&&board[i][j]<='9')

arr[board[i][j]-'0']++;

}

if((i+1)%3==0){

for(k=0;k<10;k++)

if(arr[k]>=2)

return false;

memset(arr,0,sizeof(arr));

}

}

return true;

}

## [374. 猜数字大小](https://leetcode-cn.com/problems/guess-number-higher-or-lower/)

### 1 原题目

我们正在玩一个猜数字游戏。 游戏规则如下：

我从 1 到 n 选择一个数字。 你需要猜我选择了哪个数字。

每次你猜错了，我会告诉你这个数字是大了还是小了。

你调用一个预先定义好的接口 guess(int num)，它会返回 3 个可能的结果（-1，1 或 0）：

-1 : 我的数字比较小

1 : 我的数字比较大

0 : 恭喜！你猜对了！

**示例 :**

输入: n = 10, pick = 6

输出: 6

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

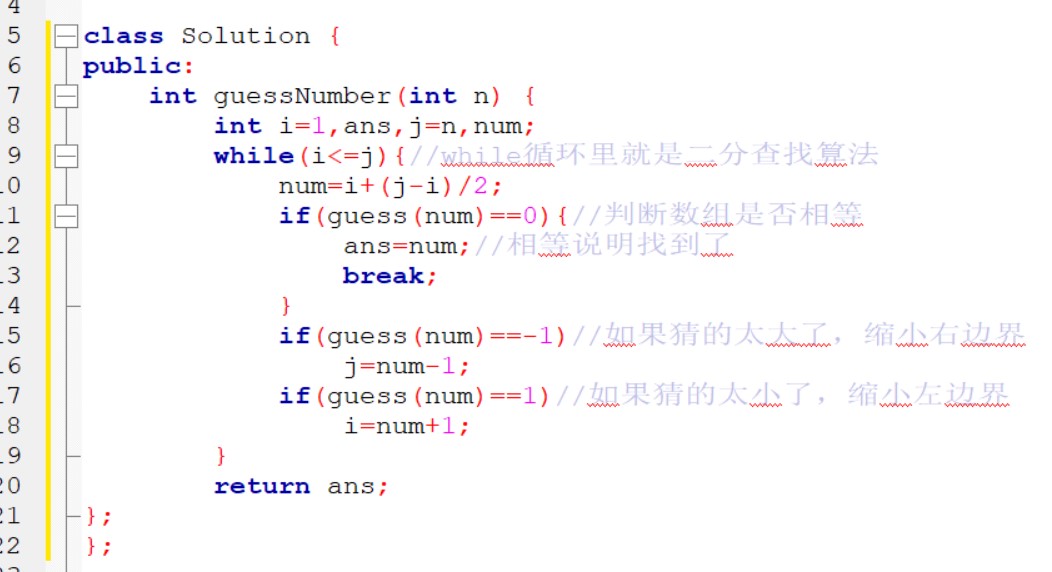
二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

根据题意所给的条件来猜数字，如果用for循环暴力解的话，会超时，所以用到二分查找这个算法，这样会加快查找的速度。

本人代码：



## [375. 猜数字大小 II](https://leetcode-cn.com/problems/guess-number-higher-or-lower-ii/)

### 1 原题目

我们正在玩一个猜数游戏，游戏规则如下：

我从 1 到 n 之间选择一个数字，你来猜我选了哪个数字。

每次你猜错了，我都会告诉你，我选的数字比你的大了或者小了。

然而，当你猜了数字 x 并且猜错了的时候，你需要支付金额为 x 的现金。直到你猜到我选的数字，你才算赢得了这个游戏。

**示例:**

n = 10, 我选择了8.

第一轮: 你猜我选择的数字是5，我会告诉你，我的数字更大一些，然后你需要支付5块。

第二轮: 你猜是7，我告诉你，我的数字更大一些，你支付7块。

第三轮: 你猜是9，我告诉你，我的数字更小一些，你支付9块。

游戏结束。8 就是我选的数字。

你最终要支付 5 + 7 + 9 = 21 块钱。

给定 n ≥ 1，计算你至少需要拥有多少现金才能确保你能赢得这个游戏。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（java已过）

可以参考以下两个链接的解释：

<https://www.hrwhisper.me/leetcode-guess-number-higher-lower-ii/>

<https://tryanswer.github.io/2018/06/02/n375-guess-number-higher-or-lower-ii/>

本人代码：

class Solution {

    public int getMoneyAmount(int n) {

        /\*\*

        dp[i][j]表示从[i,j]中猜出正确数字所需要的最少花费金额.(dp[i][i] = 0)

        假设在范围[i,j]中选择x, 则选择x的最少花费金额为: max(dp[i][x-1], dp[x+1][j]) + x

        用max的原因是我们要计算最坏反馈情况下的最少花费金额(选了x之后, 正确数字落在花费更高的那侧)

        初始化为(n+2)\*(n+2)数组的原因: 处理边界情况更加容易, 例如对于求解dp[1][n]时x如果等于1, 需要考虑dp[0][1](0不可能出现, dp[0][n]为0)

        而当x等于n时, 需要考虑dp[n+1][n+1](n+1也不可能出现, dp[n+1][n+1]为0)

        如何写出相应的代码更新dp矩阵, 递推式dp[i][j] = max(max(dp[i][x-1], dp[x+1][j]) + x), x~[i:j], 可以画出矩阵图协助理解, 可以发现

        dp[i][x-1]始终在dp[i][j]的左部, dp[x+1][j]始终在dp[i][j]的下部, 所以更新dp矩阵时i的次序应当遵循bottom到top的规则, j则相反, 由于

        i肯定小于等于j, 所以我们只需要遍历更新矩阵的一半即可(下半矩阵)

        \*\*/

        int[][] dp = new int[n+2][n+2];

        for(int i = n; i >= 1; i--) {

            for(int j = i; j <= n; j++) {

                if(i == j)

                    dp[i][j] = 0;

                else {

                    dp[i][j] = Integer.MAX\_VALUE;

                    for(int x = i; x <= j; x++)

                        dp[i][j] = Math.min(dp[i][j], Math.max(dp[i][x-1], dp[x+1][j]) + x);

                }

            }

        }

        return dp[1][n];

    }

}

## [35. 搜索插入位置](https://leetcode-cn.com/problems/search-insert-position/)

### 1 原题目

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

你可以假设数组中无重复元素。

**示例 1:**

输入: [1,3,5,6], 5

输出: 2

**示例 2:**

输入: [1,3,5,6], 2

输出: 1

**示例 3:**

输入: [1,3,5,6], 7

输出: 4

**示例 4:**

输入: [1,3,5,6], 0

输出: 0

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

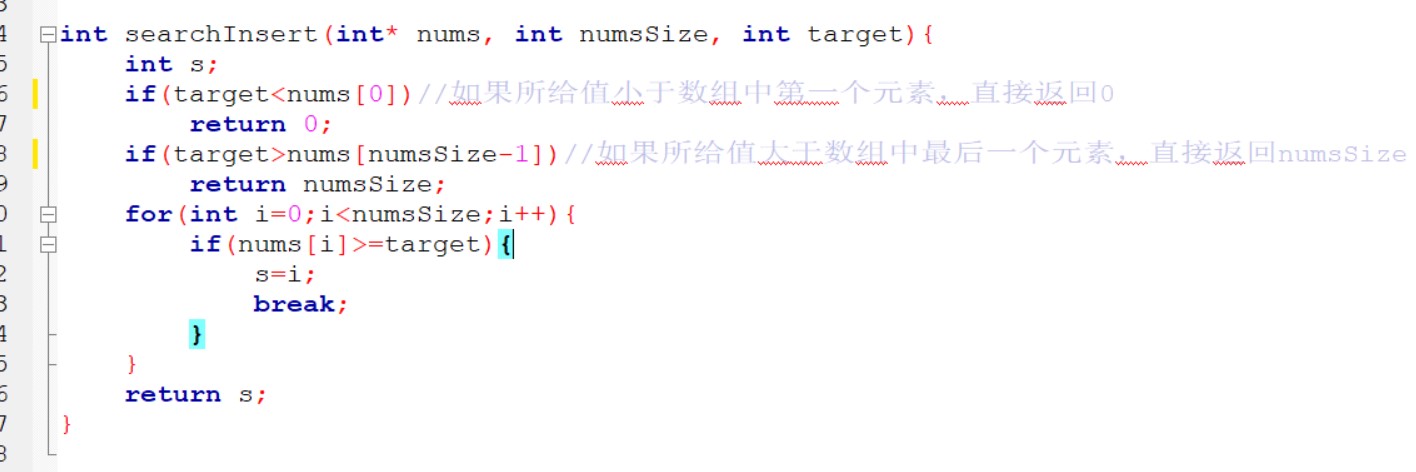
二分查找（解法二用到）

### 3 解法

#### 3.1 解法1（C语言已过）

因为数组有序，所以可以直接用for循环遍历数组来解决这个问题

本人代码：

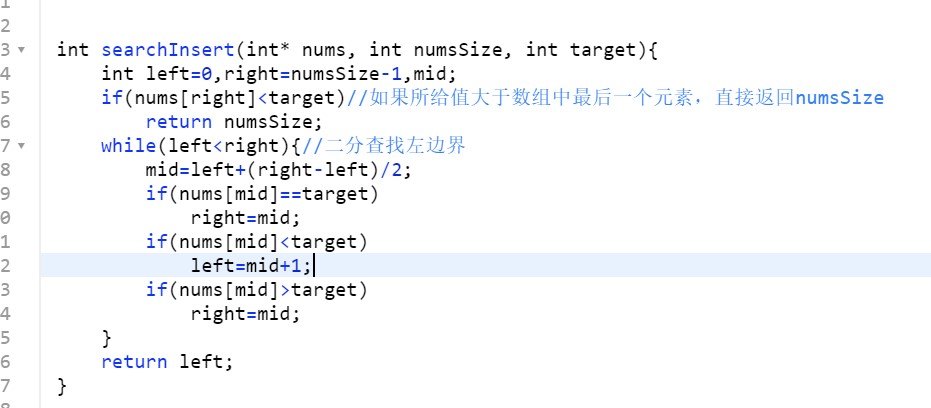


#### 3.2 解法2（C语言已过）

二分查找（查找值的左边界）



本人代码：



## [34. 在排序数组中查找元素的第一个和最后一个位置](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/)

### 1 原题目

给定一个按照升序排列的整数数组 nums，和一个目标值 target。找出给定目标值在数组中的开始位置和结束位置。

你的算法时间复杂度必须是 O(log n) 级别。

如果数组中不存在目标值，返回 [-1, -1]。

**示例 1:**

输入: nums = [5,7,7,8,8,10], target = 8

输出: [3,4]

**示例 2:**

输入: nums = [5,7,7,8,8,10], target = 6

输出: [-1,-1]

### 2 基础知识

#### 2.1数据结构

无。

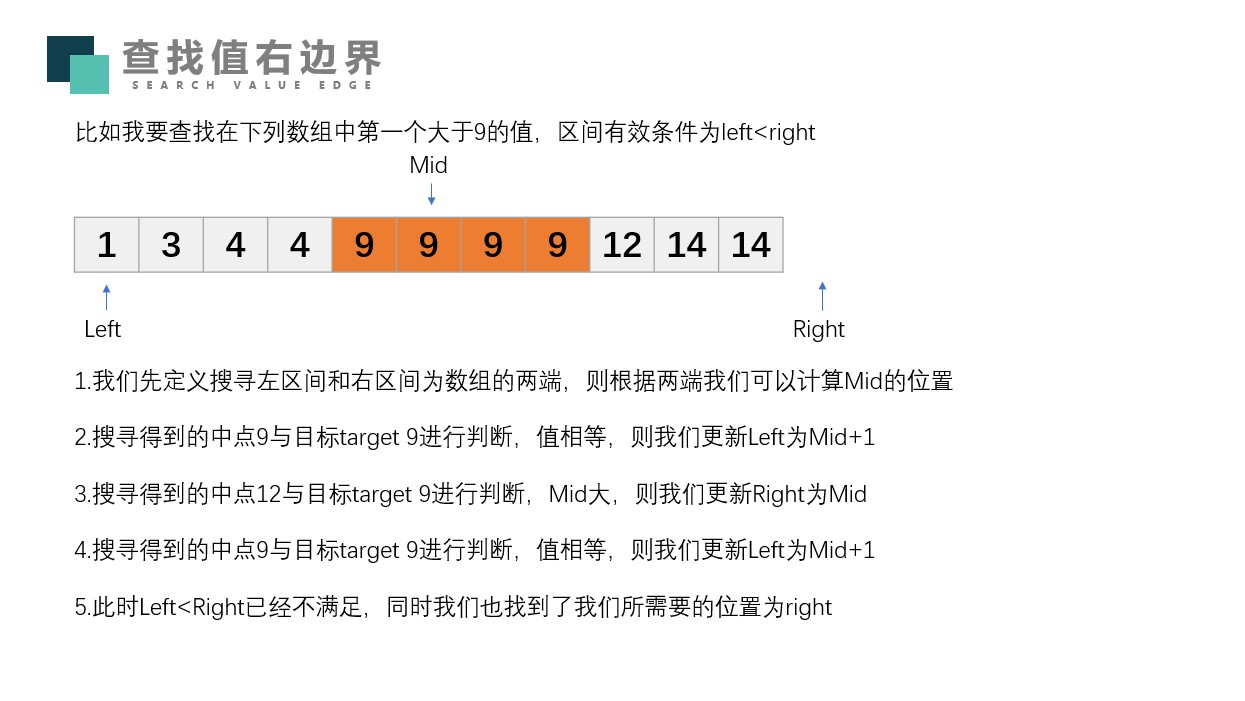
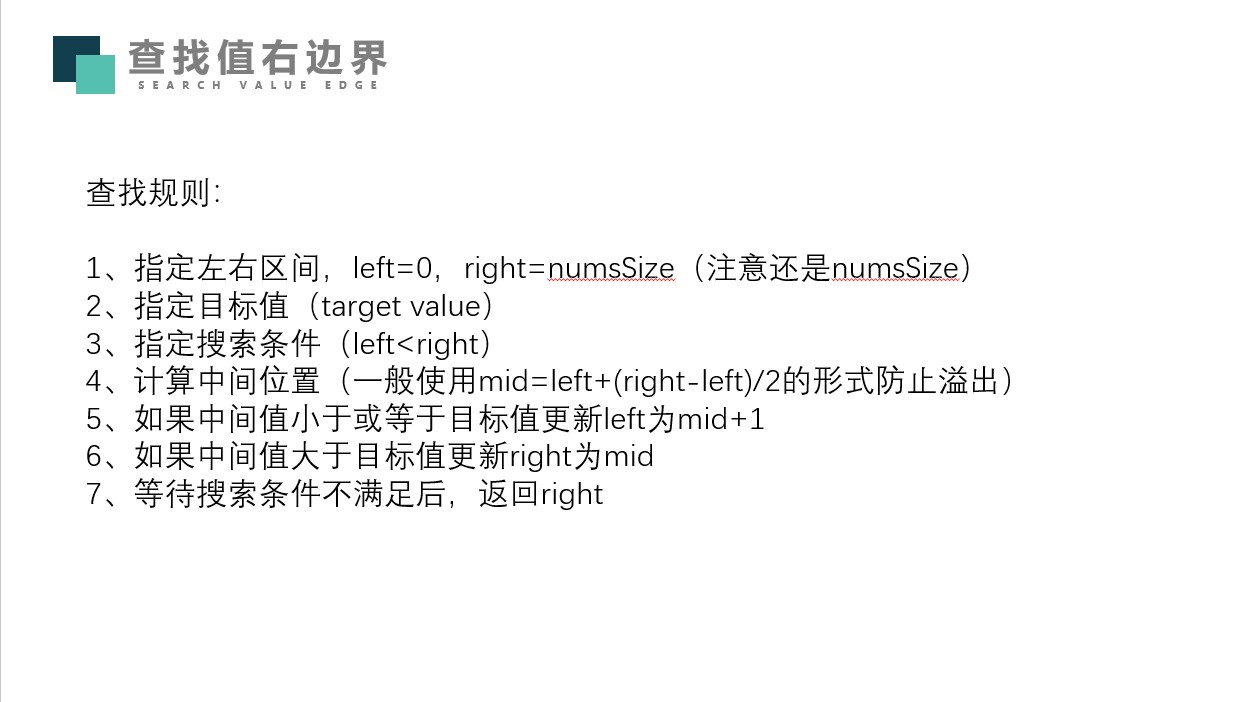
#### 2.2算法

二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

用二分查找算法来做（查找值的左边界和右边界），将下面的图片和我的代码结合来看更有助于理解。



本人代码：

int\* searchRange(int\* nums, int numsSize, int target, int\* returnSize){

int \*arr=(int \*)malloc(sizeof(int)\*2);//开辟数组

\*returnSize=2;//数组大小为2

int left=0,right=numsSize;

while(left<right){//查找值左边界

int mid=left+(right-left)/2;

if(nums[mid]>=target)

right=mid;

else

left=mid+1;

}

if(left>=0&&left<numsSize&&nums[left]==target)//限定条件

arr[0]=left;

else

arr[0]=-1;

left=0;

right=numsSize;

while(left<right){//查找值右边界

int mid=left+(right-left)/2;

if(nums[mid]<=target)

left=mid+1;

else

right=mid;

}

if(right-1>=0&&right-1<numsSize&&nums[right-1]==target)//因为右边界找出来的是第一个大于所给值的序号，所以减1

arr[1]=right-1;

else

arr[1]=-1;

return arr;

}

## [153. 寻找旋转排序数组中的最小值](https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array/)

### 1 原题目

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。

请找出其中最小的元素。

你可以假设数组中不存在重复元素。

**示例 1:**

输入: [3,4,5,1,2]

输出: 1

**示例 2:**

输入: [4,5,6,7,0,1,2]

输出: 0

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

通过二分查找来找到最小值

本人代码：

int findMin(int\* nums, int numsSize){

int mid;

int left=0,right=numsSize-1;

while(left<=right){//二分查找

mid=left+(right-left)/2;

if(mid+1<numsSize&&nums[mid]>nums[mid+1]){//判断是否出现了最小值

mid=mid+1;

break;

}

else{

if(nums[mid]<nums[left])

right=mid-1;

else

left=mid+1;

}

}

if(mid-1>=0&&nums[mid]<nums[mid-1])//最小值在中间时，返回序号为mid的那个值

return nums[mid];

return nums[0];//数组也有可能就是一个升序的，所以返回首元素

}

## [154. 寻找旋转排序数组中的最小值 II](https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array-ii/)

### 1 原题目

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。

请找出其中最小的元素。

注意数组中可能存在重复的元素。

**示例 1：**

输入: [1,3,5]

输出: 1

**示例 2：**

输入: [2,2,2,0,1]

输出: 0

**说明：**

这道题是 寻找旋转排序数组中的最小值 的延伸题目。

允许重复会影响算法的时间复杂度吗？会如何影响，为什么？

### 2 基础知识

二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题和 [剑指 Offer 11. 旋转数组的最小数字](https://leetcode-cn.com/problems/xuan-zhuan-shu-zu-de-zui-xiao-shu-zi-lcof/)一模一样。

个人觉得二分查找这类的题目你就是要先画出数字的分布图（折线图，表示数字的增减趋势），然后在观察你所找的值在图中是一个什么样的位置，然后根据此图来写代码。

参考链接：<https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array-ii/solution/xun-zhao-xuan-zhuan-pai-xu-shu-zu-zhong-de-zui--16/>

本人代码：

|  |
| --- |
| class Solution {  public:      int findMin(vector<int>& nums) {          int mid;          int left = 0,right = nums.size() - 1;          //二分查找          while(left < right) {              mid = left + (right - left) / 2;              if(nums[right] > nums[mid]) {                  right = mid;              }              else if(nums[right] < nums[mid]) {                  left = mid + 1;              } else {                  right--;              }          }          return nums[left];      }  }; |

## [33. 搜索旋转排序数组](https://leetcode-cn.com/problems/search-in-rotated-sorted-array/)

### 1 原题目

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 O(log n) 级别。

**示例 1:**

输入: nums = [4,5,6,7,0,1,2], target = 0

输出: 4

**示例 2:**

输入: nums = [4,5,6,7,0,1,2], target = 3

输出: -1

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

运用两次二分查找，第一次先找到最小值，将我们所要找的这个值的区间缩小，在新区间内进行二分查找，找到所给的值。

本人代码：

int search(int\* nums, int numsSize, int target){

if(numsSize==0)//数组为空时，不用寻找

return -1;

int res=-1,mid;

int left=0,right=numsSize-1;

while(left<=right){//第一次二分查找，找出最小值

mid=left+(right-left)/2;

if(mid+1>=0&&mid+1<numsSize&&nums[mid]>nums[mid+1]){

mid=mid+1;

break;

}

else{

if(nums[mid]<nums[left])

right=mid-1;

else

left=mid+1;

}

}

if(target>=nums[0]){//划分新的区间

left=0;

right=mid;

}

if(target<nums[0]){

left=mid;

right=numsSize-1;

}

while(left<=right){//第二次二分查找

mid=left+(right-left)/2;

if(nums[mid]==target){

res=mid;

break;

}

if(nums[mid]>target)

right=mid-1;

if(nums[mid]<target)

left=mid+1;

}

return res;

}

## [162. 寻找峰值](https://leetcode-cn.com/problems/find-peak-element/)

### 1 原题目

峰值元素是指其值大于左右相邻值的元素。

给定一个输入数组 nums，其中 nums[i] ≠ nums[i+1]，找到峰值元素并返回其索引。

数组可能包含多个峰值，在这种情况下，返回任何一个峰值所在位置即可。

你可以假设 nums[-1] = nums[n] = -∞。

**示例 1:**

输入: nums = [1,2,3,1]

输出: 2

解释: 3 是峰值元素，你的函数应该返回其索引 2。

**示例 2:**

输入: nums = [1,2,1,3,5,6,4]

输出: 1 或 5

解释: 你的函数可以返回索引 1，其峰值元素为 2；

  或者返回索引 5， 其峰值元素为 6。

**说明:**

你的解法应该是 O(logN) 时间复杂度的。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

二分查找的基础题，没啥说的。

本人代码：

|  |
| --- |
| class Solution {  public:      int findPeakElement(vector<int>& nums) {          int left = 0,right = nums.size() - 1,mid;          while(left < right) {              mid = left + (right - left) / 2;              if(nums[mid] < nums[mid + 1])                  left = mid + 1;              else                  right = mid;          }          return left;      }  }; |

## [1095. 山脉数组中查找目标值](https://leetcode-cn.com/problems/find-in-mountain-array/)

### 1 原题目

（这是一个 交互式问题 ）

给你一个 山脉数组 mountainArr，请你返回能够使得 mountainArr.get(index) 等于 target 最小 的下标 index 值。

如果不存在这样的下标 index，就请返回 -1。

何为山脉数组？如果数组 A 是一个山脉数组的话，那它满足如下条件：

**首先，**A.length >= 3

**其次，**在 0 < i < A.length - 1 条件下，存在 i 使得：

A[0] < A[1] < ... A[i-1] < A[i]

A[i] > A[i+1] > ... > A[A.length - 1]

你将 不能直接访问该山脉数组，必须通过 MountainArray 接口来获取数据：

MountainArray.get(k) - 会返回数组中索引为k 的元素（下标从 0 开始）

MountainArray.length() - 会返回该数组的长度

**注意：**

对 MountainArray.get 发起超过 100 次调用的提交将被视为错误答案。此外，任何试图规避判题系统的解决方案都将会导致比赛资格被取消。

为了帮助大家更好地理解交互式问题，我们准备了一个样例 “答案”：[https://leetcode-cn.com/playground/RKhe3ave](https://leetcode-cn.com/playground/RKhe3ave/)，请注意这 不是一个正确答案。

**示例 1：**

输入：array = [1,2,3,4,5,3,1], target = 3

输出：2

解释：3 在数组中出现了两次，下标分别为 2 和 5，我们返回最小的下标 2。

**示例 2：**

输入：array = [0,1,2,4,2,1], target = 3

输出：-1

解释：3 在数组中没有出现，返回 -1。

**提示：**

3 <= mountain\_arr.length() <= 10000

0 <= target <= 10^9

0 <= mountain\_arr.get(index) <= 10^9

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

二分查找（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

因为所给的是山脉数组，我们可以先利用二分查找找到这个山脉数组的峰值。因为题目所查找的target有可能出现两次，左右各一次，题目要求返回下标小的，我们利用峰值将数组分成两部分（左半部分升序，右半部分降序），所以我们可以先二分查找左半部分，如果没结果再查找右半部分。bisect函数之所以这么写是因为左右一个升序一个降序，这样写函数就是将左右全变成升序而已（只写一次函数，哈哈哈哈哈，你要是勤快，也可以左右各写一个二分查找的函数）。

本人代码：

|  |
| --- |
| /\*\*   \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \* // This is the MountainArray's API interface.   \* // You should not implement it, or speculate about its implementation   \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \*   \* int get(MountainArray \*, int index);   \* int length(MountainArray \*);   \*/  int bisect(MountainArray\* mountainArr, int left, int right, int target, int reverse) {      //这里使用一个reverse变量搞定升序降序，当升序 reverse=1,否则 reverse=-1      target \*= reverse;  // 全部转为升序查找      while (left <= right) {          int mid = left + (right - left) / 2;          int res = reverse \* get(mountainArr, mid);          if (res == target) {              return mid;          } else if (res < target) {              left = mid + 1;          } else              right = mid - 1;      }      return -1;  }  int findInMountainArray(int target, MountainArray\* mountainArr) {      // 1.搜索山顶下标，一定存在，因此循环结束，left或right就是山顶下标      int left = 0, right = length(mountainArr) - 1, mid;      while (left <= right) {          mid = left + (right - left) / 2;          if (get(mountainArr, mid) < get(mountainArr, mid + 1)) {              // 一定在山顶左边              left = mid + 1;          } else              right = mid - 1;      }      // 2.左右搜索      int idx = bisect(mountainArr, 0, left, target, 1);//注意左边原本升序传入 1      if (idx != -1)          return idx;      return bisect(mountainArr, left + 1, length(mountainArr) - 1, target, -1);//右边降序传入-1  } |

## [410. 分割数组的最大值](https://leetcode-cn.com/problems/split-array-largest-sum/)

### 1 原题目

给定一个非负整数数组和一个整数 m，你需要将这个数组分成 m 个非空的连续子数组。设计一个算法使得这 m 个子数组各自和的最大值最小。

**注意:**

数组长度 n 满足以下条件:

1 ≤ n ≤ 1000

1 ≤ m ≤ min(50, n)

**示例:**

输入:

nums = [7,2,5,10,8]

m = 2

输出:

18

解释:

一共有四种方法将nums分割为2个子数组。

其中最好的方式是将其分为[7,2,5] 和 [10,8]，

因为此时这两个子数组各自的和的最大值为18，在所有情况中最小。

### 2 基础知识

二分查找+贪心（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

我们知道，我们所求的值一定是落在 [max(nums), sum(nums)] 之间，所以我们可以在此区间进行二分查找，好了，关键的地方来了，每次二分查找的中值mid就是我们假设此时划分数组中数组的最大值，然后计算数组和最大值不大于mid对应的子数组个数 cnt(这个是关键！)如果 cnt>m，说明划分的子数组多了，即我们找到的 mid 偏小，故 left=mid+1;否则，说明划分的子数组少了，即 mid 偏大(或者正好就是目标值)，故right=mid。

参考链接：<https://leetcode-cn.com/problems/split-array-largest-sum/solution/er-fen-cha-zhao-by-coder233-2/>

本人代码：

|  |
| --- |
| class Solution {  public:      bool check(vector<int>& nums,int m,int mid) {          //由前向后对nums进行划分，使其子数组和 <= mid，然后统计满足条件的子数组数量          int cnt = 1;          long long sum = 0;          for(int i = 0; i < nums.size(); i++) {              if(sum + nums[i] > mid) {                  cnt++;                  sum = nums[i];              }              else                  sum += nums[i];          }          return cnt <= m;      }      int splitArray(vector<int>& nums, int m) {          //最小和必然落在 [max(nums), sum(nums)] 之间          //left表示的是左区间，初始值为max(nums)          //right表示的是右区间，初始值为sum(nums)          long long left = 0,right = 0;          for(int i = 0; i < nums.size(); i++) {              right += nums[i];              left = left > nums[i] ? left : nums[i];          }          //若我们选的mid值过小，则满足条件的数量 > m，目标值应落在 [mid+1, right]          //若我们选的mid值过大，则满足条件的数量 <= m，目标值应落在 [left, mid]          while(left < right) {//二分查找              int mid = (left + right) / 2;              if(check(nums,m,mid))                  right = mid;              else                  left = mid + 1;          }          return left;      }  }; |

## [349. 两个数组的交集](https://leetcode-cn.com/problems/intersection-of-two-arrays/)

### 1 原题目

给定两个数组，编写一个函数来计算它们的交集。

**示例 1:**

输入: nums1 = [1,2,2,1], nums2 = [2,2]

输出: [2]

**示例 2:**

输入: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

输出: [9,4]

**说明:**

输出结果中的每个元素一定是唯一的。

我们可以不考虑输出结果的顺序。

### 2 基础知识

#### 2.1数据结构

哈希表（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过打表的方式记录出两个数组中短一点的那个数组中元素出现的次数，然后遍历长数组，如果元素次数大于零，就加入到交集的数组中。

本人代码：

int\* intersection(int\* nums1, int nums1Size, int\* nums2, int nums2Size, int\* returnSize){

int \*arr=(int \*)malloc(5000\*sizeof(int));//开辟一个存放交集的数组

int t=0,i,arr1[5000]={0};//数组arr1为哈希表

\*returnSize=0;

memset(arr,0,sizeof(arr));//初始化

if(nums1Size==0&&nums2Size==0)

return arr;

if(nums1Size<=nums2Size){//判断短的数组是哪一个

for(i=0;i<nums1Size;i++)//遍历短数组

arr1[nums1[i]]++;//记录元素出现的次数

for(i=0;i<nums2Size;i++)//遍历长数组

if(arr1[nums2[i]]>=1){//如果元素出现次数大于等于1

arr1[nums2[i]]=0;//记录此元素数量的值置为0

arr[t++]=nums2[i];//将此元素加入到交集的数组中

}

}

else{

for(i=0;i<nums2Size;i++)

arr1[nums2[i]]++;

for(i=0;i<nums1Size;i++)

if(arr1[nums1[i]]>=1){

arr1[nums1[i]]=0;

arr[t++]=nums1[i];

}

}

\*returnSize=t;//交集数组中元素的个数

return arr;

}

## [98. 验证二叉搜索树](https://leetcode-cn.com/problems/validate-binary-search-tree/)

### 1 原题目

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

节点的左子树只包含小于当前节点的数。

节点的右子树只包含大于当前节点的数。

所有左子树和右子树自身必须也是二叉搜索树。

**示例 1:**

输入:

2

/ \

1 3

输出: true

**示例 2:**

输入:

5

/ \

1 4

  / \

  3 6

输出: false

解释: 输入为: [5,1,4,null,null,3,6]。

  根节点的值为 5 ，但是其右子节点值为 4 。

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

官方题解中的一种，跟据题意所给的搜索二叉树的条件，可以将二叉树进行中序遍历，将元素存储在数组中，如果是搜索二叉树，那么数组中的元素应该是升序的。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

static int arr[10000],j;//全局变量

void inorder(struct TreeNode\* t){//树的中序遍历

if(t){

inorder(t->left);

arr[j++]=t->val;//将元素存进数组中

inorder(t->right);

}

}

bool isValidBST(struct TreeNode\* root){

int i;

j=0;

memset(arr,0,sizeof(arr));//数组初始化

inorder(root);//中序遍历

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

if(arr[i]>=arr[i+1]){//满足此条件就说明不是二叉搜索树

return false;

}

}

return true;

}

## [111. 二叉树的最小深度](https://leetcode-cn.com/problems/minimum-depth-of-binary-tree/)

### 1 原题目

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

**说明:**叶子节点是指没有子节点的节点。

**示例:**

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回它的最小深度  2.

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

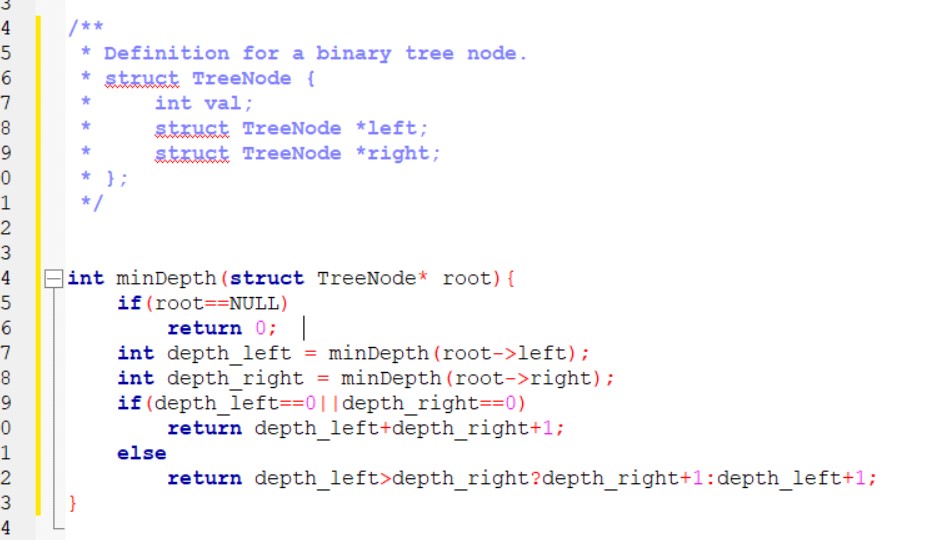
无。

### 3 解法

#### 3.1 解法1（C语言已过）

参考链接：<https://blog.csdn.net/qiki_tangmingwei/article/details/79326602>

本人代码：



## [104. 二叉树的最大深度](https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/)

### 1 原题目

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

**说明:** 叶子节点是指没有子节点的节点。

**示例：**

给定二叉树 [3,9,20,null,null,15,7]，

3

/ \

9 20

/ \

15 7

返回它的最大深度 3 。

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

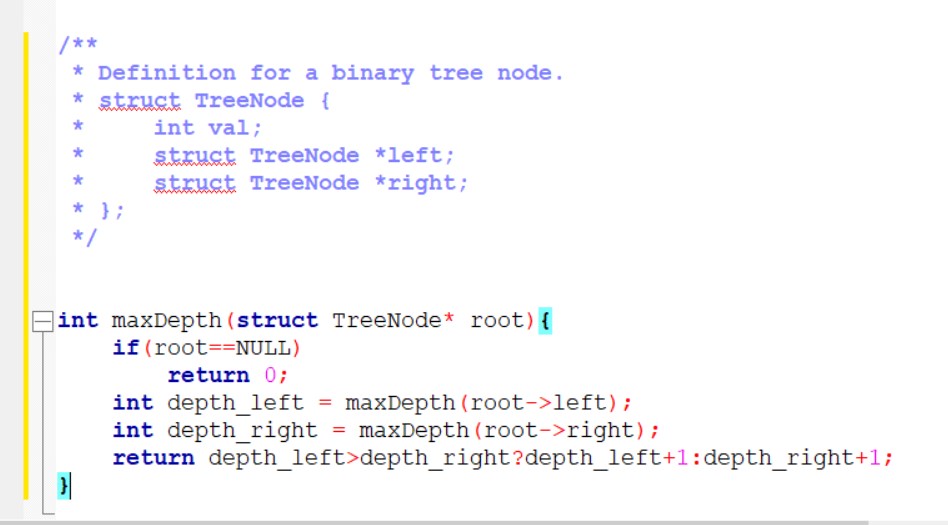
无。

### 3 解法

#### 3.1 解法1（C语言已过）

参考链接：<https://blog.csdn.net/qiki_tangmingwei/article/details/79326602>

本人代码：



## [226. 翻转二叉树](https://leetcode-cn.com/problems/invert-binary-tree/)

### 1 原题目

翻转一棵二叉树。

**示例：**

输入：

4

/ \

2 7

/ \ / \

1 3 6 9

输出：

4

/ \

7 2

/ \ / \

9 6 3 1

**备注:**

这个问题是受到 Max Howell 的 原问题 启发的 ：

谷歌：我们90％的工程师使用您编写的软件(Homebrew)，但是您却无法在面试时在白板上写出翻转二叉树这道题，这太糟糕了。

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

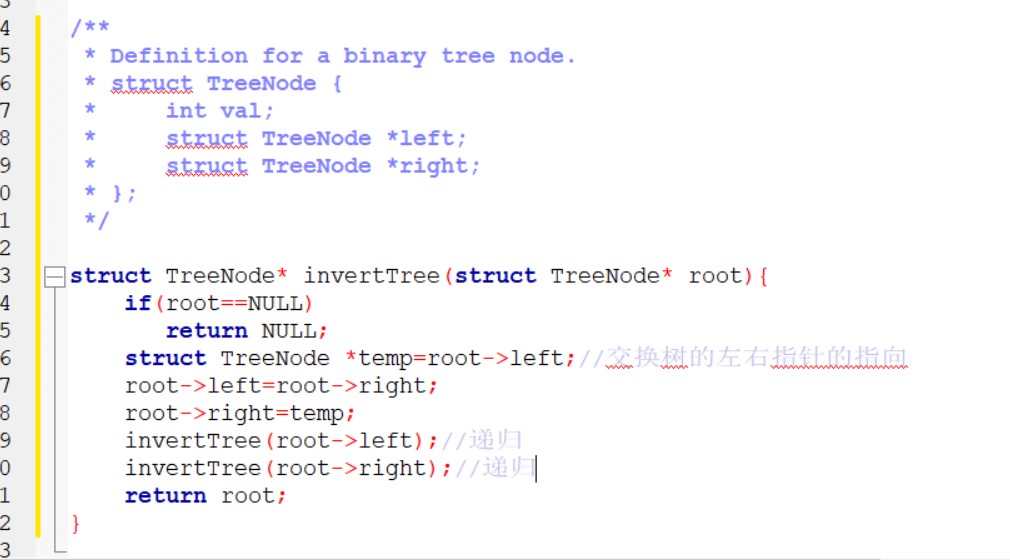
无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过将树的左右指针的指向进行更改来达到翻转二叉树的目的

本人代码：



## [230. 二叉搜索树中第K小的元素](https://leetcode-cn.com/problems/kth-smallest-element-in-a-bst/)

### 1 原题目

给定一个二叉搜索树，编写一个函数 kthSmallest 来查找其中第 k 个最小的元素。

**说明：**

你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。

**示例 1:**

输入: root = [3,1,4,null,2], k = 1

3

/ \

1 4

\

  2

输出: 1

**示例 2:**

输入: root = [5,3,6,2,4,null,null,1], k = 3

5

/ \

3 6

/ \

2 4

/

1

输出: 3

**进阶：**

如果二叉搜索树经常被修改（插入/删除操作）并且你需要频繁地查找第 k 小的值，你将如何优化 kthSmallest 函数？

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

根据二叉搜索树的性质可知，将此树进行中序遍历可得到一个升序排列的数组，所以直接将树进行中序遍历，输出第k个元素

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

static int arr[100000],j;//全局变量

void inorder(struct TreeNode\* t){//树的中序遍历

if(t){

inorder(t->left);

arr[j++]=t->val;//将元素存进数组中

inorder(t->right);

}

}

int kthSmallest(struct TreeNode\* root, int k){

j=1;

memset(arr,0,sizeof(arr));//数组初始化

inorder(root);//中序遍历

return arr[k];//返回第k小的元素

}

## [1028. 从先序遍历还原二叉树](https://leetcode-cn.com/problems/recover-a-tree-from-preorder-traversal/)

### 1 原题目

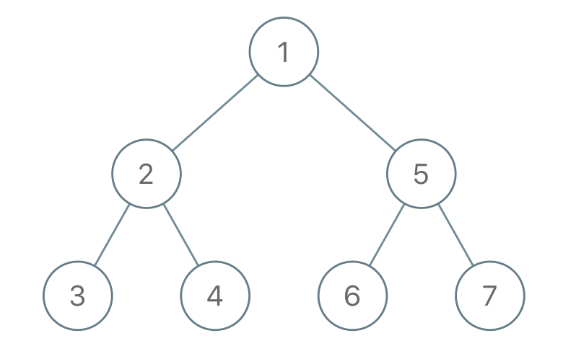
我们从二叉树的根节点 root 开始进行深度优先搜索。

在遍历中的每个节点处，我们输出 D 条短划线（其中 D 是该节点的深度），然后输出该节点的值。（如果节点的深度为 D，则其直接子节点的深度为 D + 1。根节点的深度为 0）。

如果节点只有一个子节点，那么保证该子节点为左子节点。

给出遍历输出 S，还原树并返回其根节点 root。

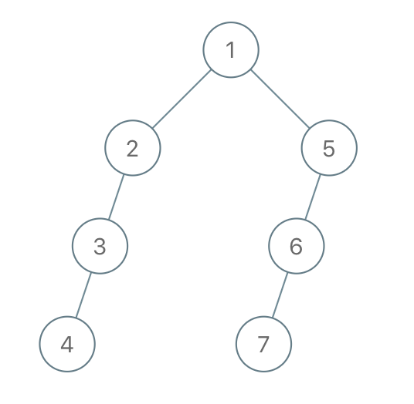
**示例 1：**



输入："1-2--3--4-5--6--7"

输出：[1,2,5,3,4,6,7]

**示例 2：**



输入："1-2--3---4-5--6---7"

输出：[1,2,5,3,null,6,null,4,null,7]

**示例 3：**



输入："1-401--349---90--88"

输出：[1,401,null,349,88,90]

**提示：**

原始树中的节点数介于 1 和 1000 之间。

每个节点的值介于 1 和 10 ^ 9 之间。

### 2 基础知识

回溯（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用回溯来解决此题，因为题目所给的字符串中用’-’来表示节点所处的深度，所以我们可以利用节点深度的比较来确定是否继续递归还是回溯到上一层。

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}   \* };   \*/  class Solution {  public:      int bit;//字符串当前遍历到的下标      TreeNode\* buildtree(string& S,int num) {//num是当前节点的深度          int begin = bit,end = bit;//记录数字开始和结束的位置          while(S[end] != '-' && end < S.length())              end++;          int val = stoi(S.substr(begin, end - begin));//字符串转成数字          bit = end - 1;          TreeNode\* res = new TreeNode(val);          int l = 0;//记录下个左节点的深度          int bitt = bit;//节点深度不对回溯到上一层时下标也要回溯          for(int i = bit + 1; i < S.length(); i++) {              if(S[i] == '-')                  l++;              else {                  bit = i;                  break;              }          }          if(l == num + 1) {//下一节点深度需要是上一节点深度加1              res -> left = buildtree(S,l);          }          else {//否则回溯              bit = bitt;              res -> left = nullptr;              return res;          }          int r = 0;//记录下个右节点的深度          bitt = bit;//节点深度不对回溯到上一层时下标也要回溯          for(int i = bit + 1; i < S.length(); i++) {              if(S[i] == '-')                  r++;              else {                  bit = i;                  break;              }          }          if(r == num + 1) {//下一节点深度需要是上一节点深度加1              res -> right = buildtree(S,r);          }          else {//否则回溯              bit = bitt;              res -> right = nullptr;              return res;          }          return res;      }      TreeNode\* recoverFromPreorder(string S) {          return buildtree(S,0);      }  }; |

## [114. 二叉树展开为链表](https://leetcode-cn.com/problems/flatten-binary-tree-to-linked-list/)

### 1 原题目

给定一个二叉树，原地将它展开为一个单链表。

例如，给定二叉树

1

/ \

2 5

/ \ \

3 4 6

将其展开为：

1

\

2

\

3

\

4

\

5

\

6

### 2 基础知识

递归，先序遍历（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

就是利用先序遍历，重复一个过程：左子树连到右子树，左子树置空，右子树连到左子树最后一个节点上，利用递归重复此过程。看代码的注释。

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}   \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}   \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}   \* };   \*/  class Solution {  public:      //采用先序遍历，根左右      void flatten(TreeNode\* root) {          if(root == nullptr)              return ;          //保留当前节点右子树          TreeNode\* oldRight = root -> right;          //把当前节点左子树放到原来右子树的位置          root -> right = root -> left;          //左子树置空          root -> left = nullptr;          //找到原先左子树的最右节点          TreeNode\* rightNode = root;          while(rightNode -> right != nullptr)              rightNode = rightNode -> right;          //将原先的左子树接到rightNode的右子树上          rightNode -> right = oldRight;          //按照先序遍历的次序，递归调用左右子树，但是左子树已经指向nullptr，因此可省去          //flatten(root->left);          flatten(root -> right);      }  }; |

## [297. 二叉树的序列化与反序列化](https://leetcode-cn.com/problems/serialize-and-deserialize-binary-tree/)

### 1 原题目

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

**示例:**

你可以将以下二叉树：

1

/ \

2 3

/ \

4 5

序列化为 "[1,2,3,null,null,4,5]"

**提示:** 这与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树的格式。你并非必须采取这种方式，你也可以采用其他的方法解决这个问题。

**说明:** 不要使用类的成员 / 全局 / 静态变量来存储状态，你的序列化和反序列化算法应该是无状态的。

### 2 基础知识

递归（解法一用到）

回溯（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

序列化：将二叉树进行先序遍历，其中’,’用来分隔数字，’#’提醒此节点为空节点。

反序列化：依靠上面规则先序建立二叉树。

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}   \* };   \*/  class Codec {  public:      // Encodes a tree to a single string.      string serialize(TreeNode\* root) {          string res = "";//存储二叉树序列化的结果          if(root) {//先序遍历（根左右）              int val = root -> val;              if(val < 0) {//对负数进行处理                  res += '-';//结果中加负号                  val \*= -1;//变成正数              }              if(val == 0)//值为0的情况                  res += 0 + '0';              string temp = "";              while(val) {//将正整数转成字符串                  temp += val % 10 + '0';                  val /= 10;              }              reverse(temp.begin(),temp.end());//字符串翻转              res += temp;              res += ',';//','目的是将数字隔开              if(root -> left) {                  res += serialize(root -> left);              }              else {//'#'目的是在反序列化时告诉它此处是空节点                  res += '#';              }              if(root -> right) {                  res += serialize(root -> right);              }              else {                  res += '#';              }          }          return res;      }      // Decodes your encoded data to tree.      TreeNode\* deserialize(string data) {          TreeNode\* res = nullptr;          int temp = buildtree(&res,data,0);          return res;      }      //先序建立二叉树      int buildtree(TreeNode\*\* res,string& data,int i) {//i为data字符串应该从哪里开始遍历          int j,b;          for(j = i; j < data.length(); j++) {              if(data[j] == ',')                  continue;              else if(data[j] == '#') {//'#'是空节点                  \*res = nullptr;                  return j + 1;              }              else {                  int flag = 0;                  if(data[j] == '-')//是否为负数                      flag = 1;                  int begin = j + flag,end = j + flag;//记录数字的开始和结束位置                  while(end < data.length() && data[end] != ',')                      end++;                  j = end - 1;                  int val = stoi(data.substr(begin, end - begin));//将字符串转成数字                  if(flag) {                      \*res = new TreeNode(-1 \* val);                  }                  else {                      \*res = new TreeNode(val);                  }                  int a = buildtree(&(\*res)->left,data,j + 1);//递归建立左子树                  b = buildtree(&(\*res)->right,data,a);//递归建立右子树                  break;              }          }          return b;      }  };  // Your Codec object will be instantiated and called as such:  // Codec codec;  // codec.deserialize(codec.serialize(root)); |

## [105. 从前序与中序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

### 1 原题目

根据一棵树的前序遍历与中序遍历构造二叉树。

**注意:**

你可以假设树中没有重复的元素。

例如，给出

前序遍历 preorder = [3,9,20,15,7]

中序遍历 inorder = [9,3,15,20,7]

返回如下的二叉树：

3

/ \

9 20

/ \

15 7

### 2 基础知识

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

注释挺详细的，就不细说了，自己看代码和注释。这是好题啊

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}   \* };   \*/  class Solution {  public:      unordered\_map<int, int> index;      TreeNode\* Build(vector<int>& preorder, vector<int>& inorder,int prel,int prer,int inl,int inr) {          if(prel > prer)              return nullptr;          //前序遍历中的第一个节点就是根节点          int preroot = prel;          //在中序遍历中定位根节点          int inroot = index[preorder[prel]];          //得到左子树中的节点数目          int value = inroot - inl;          //先把根节点建立出来          TreeNode\* root = new TreeNode(preorder[preroot]);          //递归地构造左子树，并连接到根节点          //先序遍历中「从 左边界+1 开始的 value」个元素就对应了中序遍历中「从 左边界 开始到 根节点定位-1」的元素          root -> left = Build(preorder,inorder,prel + 1,prel + value,inl,inroot - 1);          //递归地构造右子树，并连接到根节点          //先序遍历中「从 左边界+1+左子树节点数目 开始到 右边界」的元素就对应了中序遍历中「从 根节点定位+1 到 右边界」的元素          root -> right = Build(preorder,inorder,prel + value + 1,prer,inroot + 1,inr);          return root;      }      //前序遍历的顺序是根左右，中序遍历的顺序是左根右      //前序遍历的首是根结点，这个根节点在中序遍历中正好把数组分成左右两部分对应左右子树，所以就可以递归解题      TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {          int n = inorder.size();          //构造哈希映射，帮助我们快速定位根节点          for(int i = 0; i < n; i++)              index[inorder[i]] = i;          return Build(preorder,inorder,0,n - 1,0,n - 1);      }  }; |

## [106. 从中序与后序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/)

### 1 原题目

根据一棵树的中序遍历与后序遍历构造二叉树。

**注意:**

你可以假设树中没有重复的元素。

例如，给出

中序遍历 inorder = [9,3,15,20,7]

后序遍历 postorder = [9,15,7,20,3]

返回如下的二叉树：

3

/ \

9 20

/ \

15 7

### 2 基础知识

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

这题不难，懂得自然懂。后序遍历根节点是最后遍历到的，中序遍历时根节点元素又会将数组分成两部分就代表的是左右子树。所以就可以递归建树了。

参考链接：https://leetcode-cn.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/solution/tu-jie-gou-zao-er-cha-shu-wei-wan-dai-xu-by-user72/

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}   \* };   \*/  class Solution {  public:      //记录中序遍历元素对应的下标      unordered\_map<int,int> mp;      //inl,inr是中序遍历数组的左右边界，pol,por是后序遍历数组的左右边界      TreeNode\* build\_Tree(vector<int>& postorder,int inl,int inr,int pol,int por) {          if (inl > inr || pol > por)              return NULL;          //后序遍历数组的最后一个元素就是当前根节点          int rootval = postorder[por];          //inbit将中序遍历分成左右两部分代表左右两个子树          int inbit = mp[postorder[por]];          TreeNode\* node = new TreeNode(rootval);          node -> left = build\_Tree(postorder,inl,inbit - 1,pol,pol + (inbit - inl) - 1);          node -> right = build\_Tree(postorder,inbit + 1,inr,pol + (inbit - inl),por - 1);          return node;      }      TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {          for (int i = 0; i < inorder.size(); i++)              mp[inorder[i]] = i;          return build\_Tree(postorder,0,inorder.size() - 1,0,postorder.size() - 1);      }  }; |

## [144. 二叉树的前序遍历](https://leetcode-cn.com/problems/binary-tree-preorder-traversal/)

### 1 原题目

给定一个二叉树，返回它的 前序 遍历。

**示例:**

输入: [1,null,2,3]

1

\

2

/

3

输出: [1,2,3]

**进阶:** 递归算法很简单，你可以通过迭代算法完成吗？

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

树的先序遍历，没什么好说的，本解法用的是递归先序遍历

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

static n,arr[5000];//全局变量

void preorder(struct TreeNode\* t){//先序遍历

if(t){

arr[n++]=t->val;

preorder(t->left);//递归

preorder(t->right);

}

}

int\* preorderTraversal(struct TreeNode\* root, int\* returnSize){

n=0;//n用来记录数组中元素的数量

memset(arr,0,sizeof(arr));//数组初始化

preorder(root);

\*returnSize=n;

return arr;

}

## [94. 二叉树的中序遍历](https://leetcode-cn.com/problems/binary-tree-inorder-traversal/)

### 1 原题目

给定一个二叉树，返回它的中序 遍历。

**示例:**

输入: [1,null,2,3]

1

\

2

/

3

输出: [1,3,2]

**进阶:** 递归算法很简单，你可以通过迭代算法完成吗？

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

树的中序遍历，没什么好说的，本解法用的是递归中序遍历

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

static n,arr[5000];//全局变量

void inorder(struct TreeNode\* t){//递归中序遍历

if(t){

inorder(t->left);

arr[n++]=t->val;

inorder(t->right);

}

}

int\* inorderTraversal(struct TreeNode\* root, int\* returnSize){

n=0;//n用来记录数组中元素的个数

memset(arr,0,sizeof(arr));//数组的初始化

inorder(root);

\*returnSize=n;

return arr;

}

## [145. 二叉树的后序遍历](https://leetcode-cn.com/problems/binary-tree-postorder-traversal/)

### 1 原题目

给定一个二叉树，返回它的 后序 遍历。

**示例:**

输入: [1,null,2,3]

1

\

2

/

3

输出: [3,2,1]

**进阶:** 递归算法很简单，你可以通过迭代算法完成吗？

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

树的后序遍历，没什么好说的，本解法用的是递归后序遍历

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

static n,arr[5000];//全局变量

void postorder(struct TreeNode\* t){//后序遍历

if(t){

postorder(t->left);//递归

postorder(t->right);

arr[n++]=t->val;

}

}

int\* postorderTraversal(struct TreeNode\* root, int\* returnSize){

n=0;//n用来记录数组中元素的数量

memset(arr,0,sizeof(arr));//数组初始化

postorder(root);

\*returnSize=n;

return arr;

}

## [102. 二叉树的层次遍历](https://leetcode-cn.com/problems/binary-tree-level-order-traversal/)

### 1 原题目

给定一个二叉树，返回其按层次遍历的节点值。 （即逐层地，从左到右访问所有节点）。

例如:

给定二叉树: [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回其层次遍历结果：

[

[3],

[9,20],

[15,7]

]

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

就是将树进行层次遍历并存储到二维数组中，具体请看代码和注释。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int maxDepth(struct TreeNode\* root){//计算树的最大深度

if(root==NULL)

return 0;

int depth\_left = maxDepth(root->left);

int depth\_right = maxDepth(root->right);

return depth\_left>depth\_right?depth\_left+1:depth\_right+1;

}

//level就是当前元素存储的行数

void levelorder(struct TreeNode \*p,int \*\*arr,int \*col,int level){

if(p==NULL)//当节点为空就返回

return ;

arr[level][col[level]++]=p->val;//将元素放进二维数组

levelorder(p->left,arr,col,level+1);//遍历左孩子，行数加1

levelorder(p->right,arr,col,level+1);//遍历右孩子，行数加1

return ;

}

int\*\* levelOrder(struct TreeNode\* root, int\* returnSize, int\*\* returnColumnSizes){

\*returnSize=maxDepth(root);//\*returnSize记录的是行数

int i;

int \*col=(int \*)malloc((\*returnSize+1)\*sizeof(int));//开辟一个记录每行有几个元素的一维数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*(\*returnSize+1));//开辟二维数组

for(i=0; i<(\*returnSize+1); i++)

arr[i]=(int \*)malloc(sizeof(int)\*500);

memset(col,0,(\*returnSize+1)\*sizeof(int));//将col数组初始化

levelorder(root,arr,col,0);//层次遍历

\*returnColumnSizes=col;

return arr;

}

## [107. 二叉树的层次遍历 II](https://leetcode-cn.com/problems/binary-tree-level-order-traversal-ii/)

### 1 原题目

给定一个二叉树，返回其节点值自底向上的层次遍历。 （即按从叶子节点所在层到根节点所在的层，逐层从左向右遍历）

例如：

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回其自底向上的层次遍历为：

[

[15,7],

[9,20],

[3]

]

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

就是将树进行层次遍历并倒着存储到二维数组中，具体请看代码和注释。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int maxDepth(struct TreeNode\* root){//计算树的最大深度

if(root==NULL)

return 0;

int depth\_left = maxDepth(root->left);

int depth\_right = maxDepth(root->right);

return depth\_left>depth\_right?depth\_left+1:depth\_right+1;

}

//level就是当前元素存储的行数

void levelorder(struct TreeNode \*p,int \*\*arr,int \*col,int level){

if(p==NULL)//当节点为空就返回

return ;

arr[level][col[level]++]=p->val;//将元素放进二维数组

levelorder(p->left,arr,col,level-1);//遍历左孩子，行数减1

levelorder(p->right,arr,col,level-1);//遍历右孩子，行数减1

return ;

}

int\*\* levelOrderBottom(struct TreeNode\* root, int\* returnSize, int\*\* returnColumnSizes){

\*returnSize=maxDepth(root);//\*returnSize记录的是行数

int i;

int \*col=(int \*)malloc((\*returnSize+1)\*sizeof(int));//开辟一个记录每行有几个元素的一维数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*(\*returnSize+1));//开辟二维数组

for(i=0; i<(\*returnSize+1); i++)

arr[i]=(int \*)malloc(sizeof(int)\*500);

memset(col,0,(\*returnSize+1)\*sizeof(int));//将col数组初始化

levelorder(root,arr,col,\*returnSize-1);//层次遍历

\*returnColumnSizes=col;

return arr;

}

## [103. 二叉树的锯齿形层次遍历](https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal/)

### 1 原题目

给定一个二叉树，返回其节点值的锯齿形层次遍历。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

例如：

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回锯齿形层次遍历如下：

[

[3],

[20,9],

[15,7]

]

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

这道题就是先将树进行层次遍历并存储到二维数组中，最后对二维数组中奇数的行进行元素的翻转就可以了，具体请看代码和注释。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int maxDepth(struct TreeNode\* root){//计算树的最大深度

if(root==NULL)

return 0;

int depth\_left = maxDepth(root->left);

int depth\_right = maxDepth(root->right);

return depth\_left>depth\_right?depth\_left+1:depth\_right+1;

}

//level就是当前元素存储的行

void levelorder(struct TreeNode \*p,int \*\*arr,int \*col,int level){

if(p==NULL)//当节点为空就返回

return ;

arr[level][col[level]++]=p->val;//将元素放进二维数组

levelorder(p->left,arr,col,level+1);//遍历左孩子，行数加1

levelorder(p->right,arr,col,level+1);//遍历右孩子，行数加1

return ;

}

int\*\* zigzagLevelOrder(struct TreeNode\* root, int\* returnSize, int\*\* returnColumnSizes){

\*returnSize=maxDepth(root);//\*returnSize记录的是行数

int i,temp,j;

int \*col=(int \*)malloc(4000\*sizeof(int));//开辟一个记录每行几个元素的一维数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*4000);//开辟二维数组

for(i=0; i<4000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*500);

memset(col,0,4000\*sizeof(int));//将col数组初始化

levelorder(root,arr,col,0);//层次遍历

\*returnColumnSizes=col;

for(i=1;i<\*returnSize;i=i+2) {//将奇数行的元素进行翻转

for(j=0;j<col[i]/2;j++) {

temp=arr[i][j];

arr[i][j]=arr[i][col[i]-1-j];

arr[i][col[i]-1-j]=temp;

}

}

return arr;

}

## [199. 二叉树的右视图](https://leetcode-cn.com/problems/binary-tree-right-side-view/)

### 1 原题目

给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

**示例:**

输入: [1,2,3,null,5,null,4]

输出: [1, 3, 4]

解释:

1 <---

/ \

2 3 <---

\ \

5 4 <---

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将二叉树进行层次遍历并将遍历的结果存在二维数组中，然后将此二维数组中每一行的最后一个元素按顺序存进一维数组中即为树的右视图

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int maxDepth(struct TreeNode\* root){//计算树的最大深度

if(root==NULL)

return 0;

int depth\_left = maxDepth(root->left);

int depth\_right = maxDepth(root->right);

return depth\_left>depth\_right?depth\_left+1:depth\_right+1;

}

//level就是当前元素存储的行数

void levelorder(struct TreeNode \*p,int \*\*arr,int \*col,int level){

if(p==NULL)//当节点为空就返回

return ;

arr[level][col[level]++]=p->val;//将元素放进二维数组

levelorder(p->left,arr,col,level+1);//遍历左孩子，行数加1

levelorder(p->right,arr,col,level+1);//遍历右孩子，行数加1

return ;

}

int\* rightSideView(struct TreeNode\* root, int\* returnSize){

int \*res=(int \*)malloc(1000\*sizeof(int));//存右视图的数组

int num=0,i,depth=maxDepth(root);

memset(res,0,sizeof(res));//数组初始化

int \*col=(int \*)malloc((depth+1)\*sizeof(int));//开辟一个记录每行有几个元素的一维数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*(depth+1));//开辟二维数组

for(i=0; i<(depth+1); i++)

arr[i]=(int \*)malloc(sizeof(int)\*500);

memset(col,0,(depth+1)\*sizeof(int));//将col数组初始化

levelorder(root,arr,col,0);//层次遍历

for(i=0;i<depth;i++)//将层次遍历后的二维数组的每一行的最后一个元素存进res数组中，即为右视图

res[num++]=arr[i][col[i]-1];

\*returnSize=num;

return res;

}

## [108. 将有序数组转换为二叉搜索树](https://leetcode-cn.com/problems/convert-sorted-array-to-binary-search-tree/)

### 1 原题目

将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。

**示例:**

给定有序数组: [-10,-3,0,5,9],

一个可能的答案是：[0,-3,9,-10,null,5]，它可以表示下面这个高度平衡二叉搜索树：

0

/ \

-3 9

/ /

-10 5

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

二叉搜索树，它或者是一棵空树，或者是具有下列性质的[二叉树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%A0%91/1602879)： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为[二叉排序树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%8E%92%E5%BA%8F%E6%A0%91/10905079" \t "_blank)。

因为已知所给得数组是按照升序排列的，所以我们可以利用递归和二分来解决这个这个问题。就是将数组从中间分开，中间值为根节点，然后对左右数组继续递归来构建左右子树。具体请看代码和注释。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

struct TreeNode\* ToBST(int \*nums,int left,int right) {

if(left>right)//左大于右的话就没有中间值，所以返回NULL

return NULL;

int mid=(left+right)/2;//二分中值

struct TreeNode \*root =(struct TreeNode \*)malloc(sizeof(struct TreeNode));

root->val=nums[mid];

root->left=ToBST(nums,left,mid-1);//mid-1是对左半部分进行递归

root->right=ToBST(nums,mid+1,right);//mid+1是对右半部分进行递归

return root;

}

struct TreeNode\* sortedArrayToBST(int\* nums, int numsSize){

return ToBST(nums,0,numsSize-1);

}

## [109. 有序链表转换二叉搜索树](https://leetcode-cn.com/problems/convert-sorted-list-to-binary-search-tree/)

### 1 原题目

给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。

**示例:**

给定的有序链表： [-10, -3, 0, 5, 9],

一个可能的答案是：[0, -3, 9, -10, null, 5], 它可以表示下面这个高度平衡二叉搜索树：

0

/ \

-3 9

/ /

-10 5

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

二叉搜索树，它或者是一棵空树，或者是具有下列性质的[二叉树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%A0%91/1602879)： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为[二叉排序树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%8E%92%E5%BA%8F%E6%A0%91/10905079" \t "_blank)。

因为已知所给得链表是按照升序排列的，我们可以将链表中的元素存在一个数组中，所以我们可以利用递归和二分来解决这个这个问题。就是将数组从中间分开，中间值为根节点，然后对左右数组继续递归来构建左右子树。具体请看代码和注释。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

int arr[100000];

struct TreeNode\* ToBST(int left,int right) {

if(left > right)

return NULL;

int mid = (left + right) / 2;//二分中值

struct TreeNode \*root = (struct TreeNode \*)malloc(sizeof(struct TreeNode));

root->val = arr[mid];

    root->left = ToBST(left,mid-1);//注意mid-1 对左半部分进行递归

    root->right = ToBST(mid+1,right);//注意mid+1 对右半部分进行递归

    return root;

}

struct TreeNode\* sortedListToBST(struct ListNode\* head){

int k = 0;

while(head != NULL){//将链表的节点值存进一个一维数组中

arr[k++] = head->val;

head = head->next;

}

return ToBST(0,k-1);

}

## [110. 平衡二叉树](https://leetcode-cn.com/problems/balanced-binary-tree/)

### 1 原题目

给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过1。

**示例 1:**

给定二叉树 [3,9,20,null,null,15,7]

3

/ \

9 20

/ \

15 7

返回 true 。

**示例 2:**

给定二叉树 [1,2,2,3,3,null,null,4,4]

1

/ \

2 2

/ \

3 3

/ \

4 4

返回 false 。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

对每一个节点的左右子树都求出最大深度，并进行相减运算，如果每个节点左右子树的最大深度相减的绝对值的结果都小于等于1，则说明是高度平衡的二叉树。

本人代码：

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     struct TreeNode \*left;

 \*     struct TreeNode \*right;

 \* };

 \*/

int maxDepth(struct TreeNode\* root){//求二叉树的最大深度

    if(root == NULL)

        return 0;

    int depth\_left = maxDepth(root -> left);

    int depth\_right = maxDepth(root -> right);

    return depth\_left>depth\_right?depth\_left+1:depth\_right+1;

}

bool isBalanced(struct TreeNode\* root){

    if(root == NULL)//当空树的情况时

        return true;

    if(abs(maxDepth(root -> left) - maxDepth(root -> right)) <= 1)//高度平衡二叉树的条件

        if(isBalanced(root -> left) == true && isBalanced(root -> right) == true)//每个节点都要满足条件

            return true;

        else

            return false;

    else

        return false;

return true;

}

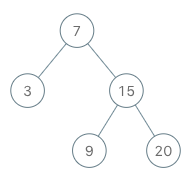
## [173. 二叉搜索树迭代器](https://leetcode-cn.com/problems/binary-search-tree-iterator/)

### 1 原题目

实现一个二叉搜索树迭代器。你将使用二叉搜索树的根节点初始化迭代器。

调用 next() 将返回二叉搜索树中的下一个最小的数。

**示例：**

****

**BSTIterator iterator = new BSTIterator(root);**

**iterator.next(); // 返回 3**

**iterator.next(); // 返回 7**

**iterator.hasNext(); // 返回 true**

**iterator.next(); // 返回 9**

**iterator.hasNext(); // 返回 true**

**iterator.next(); // 返回 15**

**iterator.hasNext(); // 返回 true**

**iterator.next(); // 返回 20**

**iterator.hasNext(); // 返回 false**

**提示：**

next() 和 hasNext() 操作的时间复杂度是 O(1)，并使用 O(h) 内存，其中 h 是树的高度。

你可以假设 next() 调用总是有效的，也就是说，当调用 next() 时，BST 中至少存在一个下一个最小的数。

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

根据二叉搜索树的性质可知，二叉搜索树中序遍历后的结果是升序的，所以将二叉搜索树中序遍历存到数组中，由此，数组中的元素也是升序排序的，剩下的就很好写了。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

typedef struct {

int top;//数组的头指针

int base;//数组的尾指针

int arr[10000];

} BSTIterator;

void inorder(struct TreeNode\* t,BSTIterator \*p){//对二叉搜索树进行中序遍历

if(t){

inorder(t->left,p);

p->arr[p->top++]=t->val;

inorder(t->right,p);

}

}

BSTIterator\* bSTIteratorCreate(struct TreeNode\* root) {

BSTIterator \*obj=(BSTIterator \*)malloc(sizeof(BSTIterator));

obj->top=0;

obj->base=0;

memset(obj->arr,0,sizeof(obj->arr));//数组初始化

inorder(root,obj);//中序遍历

return obj;

}

/\*\* @return the next smallest number \*/

int bSTIteratorNext(BSTIterator\* obj) {

int res=obj->arr[obj->base];

obj->base++;

return res;

}

/\*\* @return whether we have a next smallest number \*/

bool bSTIteratorHasNext(BSTIterator\* obj) {

if(obj->top==obj->base)

return false;

return true;

}

void bSTIteratorFree(BSTIterator\* obj) {

free(obj);

}

/\*\*

\* Your BSTIterator struct will be instantiated and called as such:

\* BSTIterator\* obj = bSTIteratorCreate(root);

\* int param\_1 = bSTIteratorNext(obj);

\* bool param\_2 = bSTIteratorHasNext(obj)

\* bSTIteratorFree(obj);

\*/

## [101. 对称二叉树](https://leetcode-cn.com/problems/symmetric-tree/)

### 1 原题目

给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树**[1,2,2,3,4,4,3]** 是对称的。

**1**

**/ \**

**2 2**

**/ \ / \**

**3 4 4 3**

但是下面这个**[1,2,2,null,3,null,3]** 则不是镜像对称的:

**1**

**/ \**

**2 2**

**\ \**

**3 3**

**说明:**

如果你可以运用递归和迭代两种方法解决这个问题，会很加分。

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

从图里可以发现，如果同时满足下面的条件，两个树互为镜像：

它们的两个根结点具有相同的值。

每个树的右子树都与另一个树的左子树镜像对称。

所以我们可以利用递归来判断这棵二叉树是不是互为镜像

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

bool iss(struct TreeNode \*left,struct TreeNode \*right){

if(left==NULL||right==NULL)

if(right==NULL&&left==NULL)

return true;

else

return false;

if(right!=NULL&&left!=NULL)

if(right->val==left->val){

if(iss(left->left,right->right)==false)//递归

return false;

if(iss(left->right,right->left)==false)//递归

return false;

}

else

return false;

return true;

}

bool isSymmetric(struct TreeNode\* root){

if(root!=NULL)

if(iss(root->left,root->right)==false)

return false;

return true;

}

## [100. 相同的树](https://leetcode-cn.com/problems/same-tree/)

### 1 原题目

给定两个二叉树，编写一个函数来检验它们是否相同。

如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

**示例 1:**

输入: 1 1

/ \ / \

2 3 2 3

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

输出: true

**示例 2:**

输入: 1 1

/ \

2 2

[1,2], [1,null,2]

输出: false

**示例 3:**

输入: 1 1

/ \ / \

2 1 1 2

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

输出: false

### 2 基础知识

#### 2.1数据结构

树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

利用递归来比较两棵二叉树是否完全一样，详情见代码。

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

bool isSameTree(struct TreeNode\* p, struct TreeNode\* q){

if((p==NULL&&q!=NULL)||(p!=NULL&&q==NULL))//有一个为空时，说明树不一样

return false;

if(p!=NULL&&q!=NULL)

if(p->val!=q->val)//结点的值不同时，说明树不一样

return false;

else{//当相同时，继续进行判断

if(isSameTree(p->left,q->left)==false)

return false;

if(isSameTree(p->right,q->right)==false)

return false;

}

return true;

}

## [116. 填充每个节点的下一个右侧节点指针](https://leetcode-cn.com/problems/populating-next-right-pointers-in-each-node/)

### 1 原题目

给定一个完美二叉树，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：

struct Node {

int val;

Node \*left;

Node \*right;

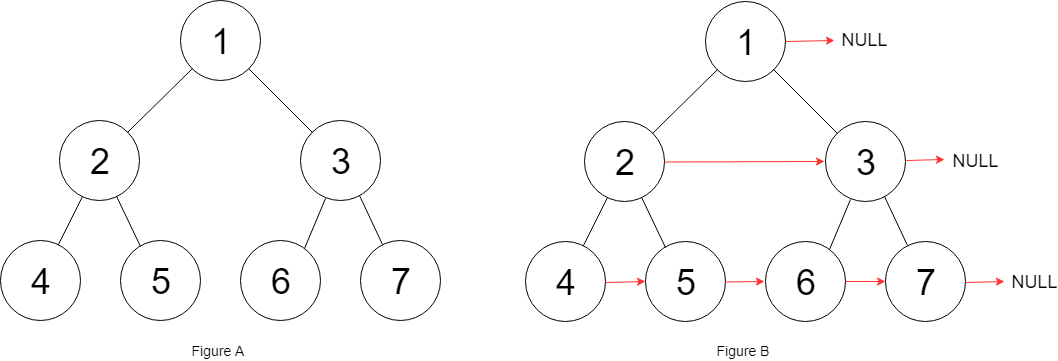
Node \*next;

}

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。

**示例：**

****

输入：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null,"right":null,"val":4},"next":null,"right":{"$id":"4","left":null,"next":null,"right":null,"val":5},"val":2},"next":null,"right":{"$id":"5","left":{"$id":"6","left":null,"next":null,"right":null,"val":6},"next":null,"right":{"$id":"7","left":null,"next":null,"right":null,"val":7},"val":3},"val":1}

输出：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":{"$id":"4","left":null,"next":{"$id":"5","left":null,"next":{"$id":"6","left":null,"next":null,"right":null,"val":7},"right":null,"val":6},"right":null,"val":5},"right":null,"val":4},"next":{"$id":"7","left":{"$ref":"5"},"next":null,"right":{"$ref":"6"},"val":3},"right":{"$ref":"4"},"val":2},"next":null,"right":{"$ref":"7"},"val":1}

解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next 指针，以指向其下一个右侧节点，如图 B 所示。

**提示：**

你只能使用常量级额外空间。

使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

利用递归的思想来解决。具体看代码和注释

本人代码：

/\*

// Definition for a Node.

class Node {

public:

int val;

Node\* left;

Node\* right;

Node\* next;

Node() {}

Node(int \_val, Node\* \_left, Node\* \_right, Node\* \_next) {

val = \_val;

left = \_left;

right = \_right;

next = \_next;

}

};

\*/

void next(Node \*root) {

if(root==NULL||root->left==NULL)//返回的条件

return ;

root->left->next=root->right;//同一个根左边的节点指向右边的节点

if(root->next!=NULL)//当不是最右边的节点时

root->right->next=root->next->left;//某个根节点的右节点指向此根右边相邻根的左节点

next(root->left);//递归左子树

next(root->right);//递归右子树

return ;

}

class Solution {

public:

Node\* connect(Node\* root) {

next(root);

return root;

}

};

## [117. 填充每个节点的下一个右侧节点指针 II](https://leetcode-cn.com/problems/populating-next-right-pointers-in-each-node-ii/)

### 1 原题目

给定一个二叉树

struct Node {

int val;

Node \*left;

Node \*right;

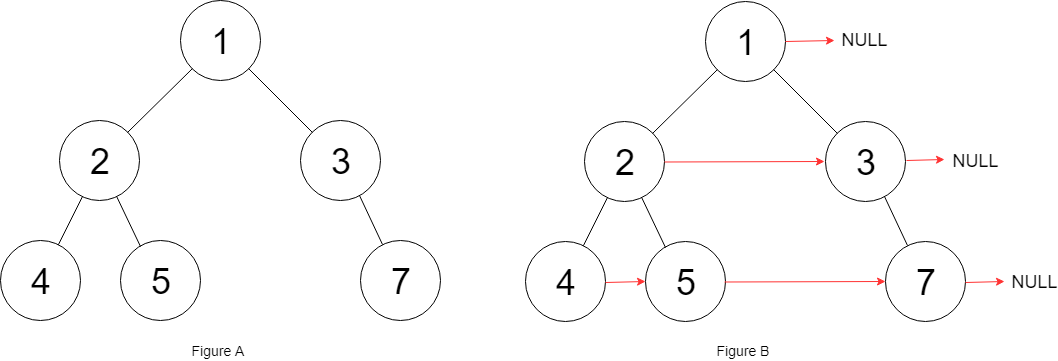
Node \*next;

}

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。

**示例：**



输入：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null,"right":null,"val":4},"next":null,"right":{"$id":"4","left":null,"next":null,"right":null,"val":5},"val":2},"next":null,"right":{"$id":"5","left":null,"next":null,"right":{"$id":"6","left":null,"next":null,"right":null,"val":7},"val":3},"val":1}

输出：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":{"$id":"4","left":null,"next":{"$id":"5","left":null,"next":null,"right":null,"val":7},"right":null,"val":5},"right":null,"val":4},"next":{"$id":"6","left":null,"next":null,"right":{"$ref":"5"},"val":3},"right":{"$ref":"4"},"val":2},"next":null,"right":{"$ref":"6"},"val":1}

解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next 指针，以指向其下一个右侧节点，如图 B 所示。

**提示：**

你只能使用常量级额外空间。

使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

利用树的层次遍历，在树的层次遍历的同时将next指针指向它的右边的节点

本人代码：

/\*

// Definition for a Node.

class Node {

public:

int val;

Node\* left;

Node\* right;

Node\* next;

Node() {}

Node(int \_val, Node\* \_left, Node\* \_right, Node\* \_next) {

val = \_val;

left = \_left;

right = \_right;

next = \_next;

}

};

\*/

struct loop//循环队列

{

Node \*base[1000];

int front;

int rear;

};

int maxDepth(Node\* root){//计算树的最大深度

if(root==NULL)

return 0;

int depth\_left = maxDepth(root->left);

int depth\_right = maxDepth(root->right);

return depth\_left>depth\_right?depth\_left+1:depth\_right+1;

}

void levelorder(Node \*root)//二叉树的层次遍历

{

Node \*p;

struct loop Q;

int maxdepth=maxDepth(root),depth=0;

Q.front=Q.rear=0;

if(root)

{

Q.base[Q.rear]=root;

Q.rear=(Q.rear+1)%1000;

Q.base[Q.rear]=NULL;//放入NULL来达到将每层分开的目的，方便记录层数

Q.rear=(Q.rear+1)%1000;

while(Q.front!=Q.rear)

{

p=Q.base[Q.front];

if(p&&Q.base[(Q.front+1)%1000]!=NULL)//p不为NULL，并且p右边存在节点时

p->next=Q.base[(Q.front+1)%1000];

if(p==NULL)//当p为NULL时

{

Q.base[Q.rear]=NULL;//加入NULL代表新的一层要开始了

Q.rear=(Q.rear+1)%1000;

depth++;//层数加1

if(depth==maxdepth)//当最后一层已经遍历结束时，就跳出while循环

break;

}

Q.front=(Q.front+1)%1000;

if(p!=NULL)//当p是树的节点时

{

if(p->left)

{

Q.base[Q.rear]=p->left;

Q.rear=(Q.rear+1)%1000;

}

if(p->right)

{

Q.base[Q.rear]=p->right;

Q.rear=(Q.rear+1)%1000;

}

}

}

}

return ;

}

class Solution {

public:

Node\* connect(Node\* root) {

levelorder(root);

return root;

}

};

## [112. 路径总和](https://leetcode-cn.com/problems/path-sum/)

### 1 原题目

给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。

**说明:** 叶子节点是指没有子节点的节点。

**示例:**

给定如下二叉树，以及目标和 sum = 22，

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

返回 true, 因为存在目标和为 22 的根节点到叶子节点的路径 5->4->11->2。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过递归来找出是否有这样一条从根节点到叶子节点的路径（具体请看代码和注释）

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

int sumpath;//计算路径总和

bool Pathsummation(struct TreeNode\* root, int sum) {

sumpath = sumpath + root->val;//总路径的和加上当前传入的节点值

if(root->left == NULL && root->right == NULL && sumpath == sum)//满足条件时，说明找到了一个路径满足题意

return true;//返回为真

if(root->left == NULL && root->right == NULL && sumpath != sum){//不满足条件时

sumpath = sumpath - root->val;//因为返回到上一级时程序接着运行，所以要总路径的和减去当前传入的节点值

return false;//返回为假

}

if(root->left != NULL)//当前节点有左孩子时

if(Pathsummation(root->left,sum) == true)

return true;

if(root->right != NULL)//当前节点有右孩子时

if(Pathsummation(root->right,sum) == true)

return true;

sumpath = sumpath - root->val;//如果发现节点的右孩子遍历完，需要继续回退到上一级

return false;

}

bool hasPathSum(struct TreeNode\* root, int sum){

sumpath=0;

if(root == NULL)//当给定的树无任何节点时

return false;

return Pathsummation(root,sum);

}

## [113. 路径总和 II](https://leetcode-cn.com/problems/path-sum-ii/)

### 1 原题目

给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。

**说明**: 叶子节点是指没有子节点的节点。

**示例:**

给定如下二叉树，以及目标和 sum = 22，

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

返回:

[

[5,4,11,2],

[5,8,4,5]

]

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过递归来找出所有从根节点到叶子节点的路径（具体请看代码和注释）

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

//sumpath为计算路径总和，k为临时数组temp的元素存储的位置，acc为行数

int sumpath,k,acc;

void Pathsummation(struct TreeNode\* root, int sum,int \*\*res,int \*col,int \*temp) {

temp[k] = root->val;//当前节点的值存入临时数组中

k=k+1;

sumpath = sumpath + root->val;//总路径的和加上当前传入的节点值

if(root->left == NULL && root->right == NULL && sumpath == sum) {//满足条件时，说明找到了一个路径满足题意

for(int i=0;i<k;i++)//将临时数组中的元素存进结果数组中

res[acc][i] = temp[i];

col[acc++] = k;//记录下此行的元素个数

k = k - 1;

sumpath = sumpath - root->val;//因为返回到上一级程序接着运行，所以要总路径的和减去当前传入的节点值

return ;

}

if(root->left == NULL && root->right == NULL && sumpath != sum){//不满足条件时

k = k - 1;

sumpath = sumpath - root->val;//因为返回到上一级程序接着运行，所以要总路径的和减去当前传入的节点值

return ;

}

if(root->left != NULL)//当前节点有左孩子时

Pathsummation(root->left,sum,res,col,temp);

if(root->right != NULL)//当前节点有右孩子时

Pathsummation(root->right,sum,res,col,temp);

k = k - 1;

sumpath = sumpath - root->val;//如果发现节点的右孩子遍历完，需要继续回退到上一级

return ;

}

int\*\* pathSum(struct TreeNode\* root, int sum, int\* returnSize, int\*\* returnColumnSizes){

sumpath = 0;

k = 0;

acc = 0;

\*returnSize = 0;

int \*col = (int \*)malloc(1000\*sizeof(int));//开辟一个记录每行元素个数的数组

int \*\*res = (int \*\*)malloc(sizeof(int\*)\*1000);//开辟二维数组用来存结果

for(int i=0; i<1000; i++)

res[i]=(int \*)malloc(sizeof(int)\*1000);

int temp[1000] = {0};

if(root == NULL)//当给定的树无任何节点时

return res;

Pathsummation(root,sum,res,col,temp);

\*returnSize = acc;

\*returnColumnSizes = col;

return res;

}

## [129. 求根到叶子节点数字之和](https://leetcode-cn.com/problems/sum-root-to-leaf-numbers/)

### 1 原题目

给定一个二叉树，它的每个结点都存放一个 0-9 的数字，每条从根到叶子节点的路径都代表一个数字。

例如，从根到叶子节点路径 1->2->3 代表数字 123。

计算从根到叶子节点生成的所有数字之和。

**说明:** 叶子节点是指没有子节点的节点。

**示例 1:**

输入: [1,2,3]

1

/ \

2 3

输出: 25

解释:

从根到叶子节点路径 1->2 代表数字 12.

从根到叶子节点路径 1->3 代表数字 13.

因此，数字总和 = 12 + 13 = 25.

**示例 2:**

输入: [4,9,0,5,1]

4

/ \

9 0

 / \

5 1

输出: 1026

解释:

从根到叶子节点路径 4->9->5 代表数字 495.

从根到叶子节点路径 4->9->1 代表数字 491.

从根到叶子节点路径 4->0 代表数字 40.

因此，数字总和 = 495 + 491 + 40 = 1026.

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过递归来找出所有从根节点到叶子节点的路径（具体请看代码和注释）

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

int sum,k;

void Pathsummation(struct TreeNode\* root,int \*temp) {

temp[k] = root->val;//当前节点的值存入临时数组中

k = k + 1;

if(root->left == NULL && root->right == NULL) {//当找到从根到叶子节点的路径

int i=0,j;

for(j = k-1;j >= 0;j--)

sum = sum + temp[j]\*pow(10,i++);//计算所有的根到叶子节点的数字之和

k = k - 1;

return ;

}

if(root->left != NULL)//当前节点有左孩子时

Pathsummation(root->left,temp);

if(root->right != NULL)//当前节点有右孩子时

Pathsummation(root->right,temp);

k = k - 1;

return ;

}

int sumNumbers(struct TreeNode\* root){

if(root == NULL)

return 0;

sum = 0;

k = 0;

int temp[1000] = {0};

Pathsummation(root,temp);

return sum;

}

## [124. 二叉树中的最大路径和](https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/)

### 1 原题目

给定一个非空二叉树，返回其最大路径和。

本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径至少包含一个节点，且不一定经过根节点。

**示例 1:**

输入: [1,2,3]

1

/ \

2 3

输出: 6

**示例 2:**

输入: [-10,9,20,null,null,15,7]

  -10

   / \

  9  20

    /  \

   15   7

输出: 42

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

可以通过递归调用来找寻最大的路径和，不断的计算节点的左右子树的路径和来选取出来一个最大值。（此题参考的链接：<https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/solution/er-cha-shu-de-zui-da-lu-jing-he-by-leetcode/>）

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

int MAX(int a,int b) {

if(a>b)

return a;

else

return b;

}

int maxsum;

int max(struct TreeNode\* root) {

if(root == NULL)

return 0;

int left\_max = MAX(max(root->left),0);//计算节点左子树的最大路径和

int right\_max = MAX(max(root->right),0);//计算节点右子树的最大路径和

int now\_max = root->val + left\_max + right\_max;//计算当前节点的最大路径和

maxsum = MAX(maxsum,now\_max);//在目前的最大值和当前节点的最大路径和二者中选取一个最大值

return root->val + MAX(left\_max,right\_max);

}

int maxPathSum(struct TreeNode\* root){

maxsum = INT\_MIN;

max(root);

return maxsum;

}

## [257. 二叉树的所有路径](https://leetcode-cn.com/problems/binary-tree-paths/)

### 1 原题目

给定一个二叉树，返回所有从根节点到叶子节点的路径。

**说明:**叶子节点是指没有子节点的节点。

**示例:**

输入:

1

/ \

2 3

\

5

输出: ["1->2->5", "1->3"]

解释: 所有根节点到叶子节点的路径为: 1->2->5, 1->3

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过递归来找出所有从根节点到叶子节点的路径（具体请看代码和注释）

C库函数sprintf的介绍链接：<https://www.runoob.com/cprogramming/c-function-sprintf.html>

本人代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int k,row;//k是临时数组temp的下标，row记录存放结果的二维数组的行数

void path(struct TreeNode\* root,char\*\* res,int\* temp) {

temp[k] = root->val;//当前节点的值存入临时数组中

k=k+1;

if(root->left == NULL && root->right == NULL) {//当找到从根到叶子节点的路径

int i=0;

for(int j=0; j<k-1; j++)//按题意的形式存进结果数组中

sprintf(res[row], "%s%d->", res[row], temp[j]);

sprintf(res[row], "%s%d", res[row], temp[k-1]);

row++;//行数加1

k = k - 1;

return ;

}

if(root->left != NULL)//当前节点有左孩子时

path(root->left,res,temp);

if(root->right != NULL)//当前节点有右孩子时

path(root->right,res,temp);

k = k - 1;

return ;

}

char \*\* binaryTreePaths(struct TreeNode\* root, int\* returnSize){

k = 0;

row = 0;

char \*\*res = (char \*\*)malloc(sizeof(char\*)\*1000);//开辟二维数组

for(int i=0; i<1000; i++){

res[i]=(char \*)malloc(sizeof(char)\*1000);

memset(res[i],0,1000\*sizeof(char));//对数组进行初始化

}

if(root == NULL) {

\*returnSize = 0;

return res;

}

int temp[1000] = {0};

path(root,res,temp);

\*returnSize = row;

return res;

}

## [99. 恢复二叉搜索树](https://leetcode-cn.com/problems/recover-binary-search-tree/)

### 1 原题目

二叉搜索树中的两个节点被错误地交换。

请在不改变其结构的情况下，恢复这棵树。

**示例 1:**

输入: [1,3,null,null,2]

  1

  /

 3

  \

  2

输出: [3,1,null,null,2]

  3

  /

 1

  \

  2

**示例 2:**

输入: [3,1,4,null,null,2]

3

/ \

1 4

  /

  2

输出: [2,1,4,null,null,3]

2

/ \

1 4

  /

 3

**进阶:**

使用 O(n) 空间复杂度的解法很容易实现。

你能想出一个只使用常数空间的解决方案吗？

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

因为题目中告诉我们只有两个节点被错误的交换了。又因为二叉搜索树的中序遍历结果是一个升序排列的。所以我们可以用一个base数组存储二叉树的节点，arr数组存储节点值。然后我们对arr进行升序排序。最后base数组中的节点值与arr数组中的值进行比较，如果值不相同，记录下标的值，最后将这两个节点的值交换。

本人代码：

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     struct TreeNode \*left;

 \*     struct TreeNode \*right;

 \* };

 \*/

struct TreeNode \*base[1000];

int n;

long long int arr[1000];

int cmp(const void \*a,const void \*b) {//从小到大排序

    const long long int p1 = \*(const long long int\*)a;

    const long long int p2 = \*(const long long int\*)b;

    if (p1 < p2)

        return -1;

    else if (p1 > p2)

        return 1;

    return 0;

}

void inorder(struct TreeNode\* t){//递归中序遍历

    if(t){

        inorder(t->left);

        base[n] = t;//存当前节点

        arr[n++] = t -> val;//存当前节点的值

        inorder(t->right);

    }

}

void recoverTree(struct TreeNode\* root){

    n = 0;

    inorder(root);//中序遍历

    qsort(arr,n,sizeof(arr[0]),cmp);//排序

    int i,temp,num = 0;//num是为了记录找到第几个不同的元素

    for(i = 0; i < n; i++) {

        if(arr[i] != base[i] -> val) {//寻找不同的元素

            if(num == 0) {//当找到第一个不同的元素时

                temp = i;//记录下来这个下标

            }

            num++;

            if(num == 2)//当找到第二个不同的元素时

                break;

        }

    }

    num = base[temp] -> val;//交换两个节点的值

    base[temp] -> val = base[i] -> val;

    base[i] -> val = num;

}

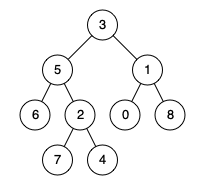
## [236. 二叉树的最近公共祖先](https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree/)

### 1 原题目

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

例如，给定如下二叉树:  root = [3,5,1,6,2,0,8,null,null,7,4]



**示例 1:**

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

输出: 3

解释: 节点 5 和节点 1 的最近公共祖先是节点 3。

**示例 2:**

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

输出: 5

解释: 节点 5 和节点 4 的最近公共祖先是节点 5。因为根据定义最近公共祖先节点可以为节点本身。

**说明:**

所有节点的值都是唯一的。

p、q 为不同节点且均存在于给定的二叉树中。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

递归（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

思路: 对每个节点root, 在其两棵子树上求p和q的公共祖先. 假设在左子树上求得公共祖先L, 在左子树上求得公共祖先R. 分情况讨论.

当 L == NULL 且 R == NULL 时, 说明p和q没有出现在root子树上. 返回空.

当 L == NULL 且 R != NULL 时, 说明p和q的公共祖先出现在右子树上. 返回R.

当 L != NULL 且 R == NULL 时, 说明p和q的公共祖先出现在左子树上. 返回L.

当 L != NULL 且 R != NULL 时, 说明p和q分别出现在左右子树上. 那么, root就是公共祖先了.

参考链接：<https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree/solution/hou-xu-bian-li-fa-by-jason-2/>

本人代码：

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     struct TreeNode \*left;

 \*     struct TreeNode \*right;

 \* };

 \*/

struct TreeNode\* lowestCommonAncestor(struct TreeNode\* root, struct TreeNode\* p, struct TreeNode\* q) {

    //如果二叉树的根节点是p,q,NULL三者其中的一种,则返回root: 1.p或者q是root,则两者的最近公共祖先是root 2.root是空的,即二叉树是空树,返回NULL,也是返回root

    if(root == NULL || root == p || root == q)

        return root;

    struct TreeNode \*left = lowestCommonAncestor(root -> left,p,q);//去左子树看是否可以访问到p或q,并把结果返回给left

    struct TreeNode \*right = lowestCommonAncestor(root -> right,p,q);//去右子树看是否可以访问到p或q,并把结果返回给right

    if(left != NULL && right != NULL)//当 left != NULL 且 right != NULL 时, 说明p和q分别出现在左右子树上. 那么, root就是公共祖先了

        return root;

    else if(left == NULL && right != NULL)//当 left == NULL 且 right != NULL 时, 说明p和q的公共祖先出现在右子树上. 返回right.

        return right;

    else if(right == NULL && left != NULL)//当 left != NULL 且 right == NULL 时, 说明p和q的公共祖先出现在左子树上. 返回left.

        return left;

    return ;

}

## [1367. 二叉树中的列表](https://leetcode-cn.com/problems/linked-list-in-binary-tree/)

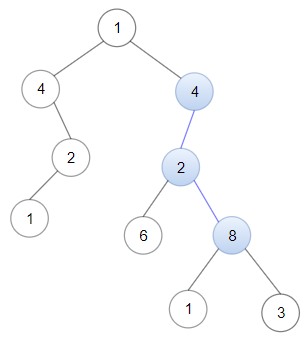
### 1 原题目

给你一棵以 root 为根的二叉树和一个 head 为第一个节点的链表。

如果在二叉树中，存在一条一直向下的路径，且每个点的数值恰好一一对应以 head 为首的链表中每个节点的值，那么请你返回 True ，否则返回 False 。

一直向下的路径的意思是：从树中某个节点开始，一直连续向下的路径。

**示例 1：**

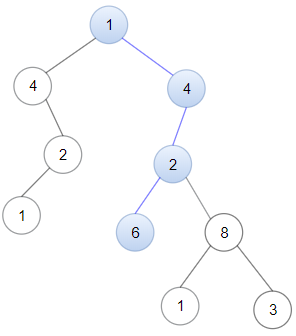


输入：head = [4,2,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,3]

输出：true

解释：树中蓝色的节点构成了与链表对应的子路径。

**示例 2：**



输入：head = [1,4,2,6], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,3]

输出：true

**示例 3：**

输入：head = [1,4,2,6,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,3]

输出：false

解释：二叉树中不存在一一对应链表的路径。

**提示：**

二叉树和链表中的每个节点的值都满足 1 <= node.val <= 100 。

链表包含的节点数目在 1 到 100 之间。

二叉树包含的节点数目在 1 到 2500 之间。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

递归算法，直接一直递归下去找列表就可以。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     struct TreeNode \*left;

 \*     struct TreeNode \*right;

 \* };

 \*/

bool judge(struct ListNode\* head, struct TreeNode\* root) {

    if(head == NULL)//列表为空时返回真

        return true;

    if(root == NULL || head -> val != root -> val)//根为空时或者值不等时返回假

        return false;

    return judge(head -> next,root -> left) || judge(head -> next,root -> right);//递归判断左右子树中的值和列表中的值是否相等

}

bool isSubPath(struct ListNode\* head, struct TreeNode\* root){

    if(head == NULL)//列表为空时返回真

        return true;

    if(root == NULL)//根为空时返回假

        return false;

    if(isSubPath(head,root -> left) == true)//左子树递归

        return true;

    if(isSubPath(head,root -> right) == true)//右子树递归

        return true;

    if(head -> val == root -> val) {//当找到了列表首个值和树中某个节点值相等

        if(judge(head -> next,root -> left) == true)//判断左子树中的值是否和列表下一个值相等

            return true;

        if(judge(head -> next,root -> right) == true)//判断右子树中的值是否和列表下一个值相等

            return true;

    }

    return false;

}

## [543. 二叉树的直径](https://leetcode-cn.com/problems/diameter-of-binary-tree/)

### 1 原题目

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过根结点。

**示例 :**

给定二叉树

1

/ \

2 3

/ \

4 5

返回 3, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。

**注意：**两结点之间的路径长度是以它们之间边的数目表示。

### 2 基础知识

#### 2.1数据结构

二叉树（解法一用到）

#### 2.2算法

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

题目所说的最大直径就是树中任意两个节点之间的距离，所以我们可以利用递归的思想来想。首先我们知道某个节点的左右深度加起来就是这个节点的最大直径了，而且我们遍历树是从根节点往下遍历，所以说除了根节点以外的点不存在通过根节点的最大直径（如果通过根节点有最大直径，那之前在计算根节点的直径时就已经计算过了）。

本人代码：

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

 \* };

 \*/

class Solution {

public:

    int maxl = 0;

    int deep(struct TreeNode\* root) {//通过递归计算二叉树的最大深度

    if(root == NULL)

        return 0;

    int left = deep(root -> left) + 1;

    int right = deep(root -> right) + 1;

    return max(left,right);

}

    int diameterOfBinaryTree(TreeNode\* root) {

        if(root == NULL)

            return 0;

        int temp = deep(root -> left) + deep(root -> right);//计算当前节点的最大直径

        maxl = maxl > temp ? maxl : temp;//取最大值

        diameterOfBinaryTree(root -> left);//递归计算左子树的直径

        diameterOfBinaryTree(root -> right);//递归计算右子树的直径

        return maxl;

    }

};

## [95. 不同的二叉搜索树 II](https://leetcode-cn.com/problems/unique-binary-search-trees-ii/)

### 1 原题目

给定一个整数 n，生成所有由 1 ... n 为节点所组成的 二叉搜索树 。

**示例：**

输入：3

输出：

[

  [1,null,3,2],

  [3,2,null,1],

  [3,1,null,null,2],

  [2,1,3],

  [1,null,2,null,3]

]

解释：

以上的输出对应以下 5 种不同结构的二叉搜索树：

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

**提示：**

0 <= n <= 8

### 2 基础知识

递归（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

二叉搜索树的特性是：根节点的左子树的值比根节点值小，根节点的右子树的值比根节点值大。知道这个知识后，就可以利用递归来求解。

参考链接：<https://leetcode-cn.com/problems/unique-binary-search-trees-ii/solution/xiang-xi-tong-su-de-si-lu-fen-xi-duo-jie-fa-by-2-7/>

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}   \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}   \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}   \* };   \*/  class Solution {  public:      vector<TreeNode\*> createTrees(int start, int end) {          if (start > end) {              return { nullptr };          }          vector<TreeNode\*> allTrees;          // 枚举可行根节点          for (int i = start; i <= end; i++) {              // 获得所有可行的左子树集合              vector<TreeNode\*> leftTrees = createTrees(start, i - 1);              // 获得所有可行的右子树集合              vector<TreeNode\*> rightTrees = createTrees(i + 1, end);              // 从左子树集合中选出一棵左子树，从右子树集合中选出一棵右子树，拼接到根节点上              for(int j = 0; j < leftTrees.size(); j++) {                  for(int k = 0; k < rightTrees.size(); k++) {                      TreeNode\* currTree = new TreeNode(i);                      currTree->left = leftTrees[j];                      currTree->right = rightTrees[k];                      allTrees.push\_back(currTree);                  }              }          }          return allTrees;      }      vector<TreeNode\*> generateTrees(int n) {          if (n == 0) {              return {};          }          return createTrees(1, n);      }  }; |

## [887. 鸡蛋掉落](https://leetcode-cn.com/problems/super-egg-drop/)

### 1 原题目

你将获得 K 个鸡蛋，并可以使用一栋从 1 到 N  共有 N 层楼的建筑。

每个蛋的功能都是一样的，如果一个蛋碎了，你就不能再把它掉下去。

你知道存在楼层 F ，满足 0 <= F <= N 任何从高于 F 的楼层落下的鸡蛋都会碎，从 F 楼层或比它低的楼层落下的鸡蛋都不会破。

每次移动，你可以取一个鸡蛋（如果你有完整的鸡蛋）并把它从任一楼层 X 扔下（满足 1 <= X <= N）。

你的目标是确切地知道 F 的值是多少。

无论 F 的初始值如何，你确定 F 的值的最小移动次数是多少？

**示例 1：**

输入：K = 1, N = 2

输出：2

解释：

鸡蛋从 1 楼掉落。如果它碎了，我们肯定知道 F = 0 。

否则，鸡蛋从 2 楼掉落。如果它碎了，我们肯定知道 F = 1 。

如果它没碎，那么我们肯定知道 F = 2 。

因此，在最坏的情况下我们需要移动 2 次以确定 F 是多少。

**示例 2：**

输入：K = 2, N = 6

输出：3

**示例 3：**

输入：K = 3, N = 14

输出：4

**提示：**

1 <= K <= 100

1 <= N <= 10000

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

其实递归利用的就是每次做最坏的打算的思想，每次递归计算时按最坏的打算看可以从当前层一下可以加到几层（就算碎了，因为已经按最坏打算，也可以一层层试下去）。

参考链接：<https://leetcode-cn.com/problems/super-egg-drop/solution/887-by-ikaruga/>

本人代码：

int fun(int K, int T)// 计算 K 个鸡蛋，扔 T 次，可以覆盖多少个区间

{

    //if (T == 1) return 2;

    //if (K == 1) return T + 1;

    if (T == 1 || K == 1) return T + 1; // 上面两句可以合并成这一句

    return fun(K - 1, T - 1) + fun(K, T - 1);

}

int superEggDrop(int K, int N)

{

    int T = 1;// 测试机会

    while (fun(K, T) < N + 1) T++;

    return T;

}

## [10. 正则表达式匹配](https://leetcode-cn.com/problems/regular-expression-matching/)

### 1 原题目

给你一个字符串 s 和一个字符规律 p，请你来实现一个支持 '.' 和 '\*' 的正则表达式匹配。

'.' 匹配任意单个字符

'\*' 匹配零个或多个前面的那一个元素

所谓匹配，是要涵盖 整个 字符串 s的，而不是部分字符串。

**说明:**

s 可能为空，且只包含从 a-z 的小写字母。

p 可能为空，且只包含从 a-z 的小写字母，以及字符 . 和 \*。

**示例 1:**

输入:

s = "aa"

p = "a"

输出: false

解释: "a" 无法匹配 "aa" 整个字符串。

**示例 2:**

输入:

s = "aa"

p = "a\*"

输出: true

解释: 因为 '\*' 代表可以匹配零个或多个前面的那一个元素, 在这里前面的元素就是 'a'。因此，字符串 "aa" 可被视为 'a' 重复了一次。

**示例 3:**

输入:

s = "ab"

p = ".\*"

输出: true

解释: ".\*" 表示可匹配零个或多个（'\*'）任意字符（'.'）。

**示例 4:**

输入:

s = "aab"

p = "c\*a\*b"

输出: true

解释: 因为 '\*' 表示零个或多个，这里 'c' 为 0 个, 'a' 被重复一次。因此可以匹配字符串 "aab"。

**示例 5:**

输入:

s = "mississippi"

p = "mis\*is\*p\*."

输出: false

### 2 基础知识

递归（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用递归来进行正则匹配，详细看注释和代码。

本人代码：

|  |
| --- |
| class Solution {  public:      //sb表示s的下标，pb表示p的下标      bool match(string& s,int sb,string& p,int pb) {          if(pb >= p.length())//正则串超出长度判断待匹配字符串是否也超出长度              return sb >= s.length();          bool curr = sb < s.length() && ((s[sb] == p[pb]) || (p[pb] == '.'));//判断当前下标是否可以匹配          if(pb + 1 < p.length() && p[pb + 1] == '\*') {//'\*' 匹配零个或多个前面的那一个元素              //匹配0个正则串需要下标移动两位，待匹配字符串下标不移动（因为可以以匹配零个所以当前元素判断是否可以对应是无所谓的）              //匹配多个正则串下标不移动，待匹配字符串下标移动一位              return match(s,sb,p,pb + 2) || (curr && match(s,sb + 1,p,pb));          }          else {//没遇到'.'或'\*'继续判断二者的下一个下标              return curr && match(s,sb + 1,p,pb + 1);          }      }      bool isMatch(string s, string p) {          return match(s,0,p,0);      }  }; |

## [207. 课程表](https://leetcode-cn.com/problems/course-schedule/)

### 1 原题目

你这个学期必须选修 numCourse 门课程，记为 0 到 numCourse-1 。

在选修某些课程之前需要一些先修课程。 例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们：[0,1]

给定课程总量以及它们的先决条件，请你判断是否可能完成所有课程的学习？

**示例 1:**

输入: 2, [[1,0]]

输出: true

解释: 总共有 2 门课程。学习课程 1 之前，你需要完成课程 0。所以这是可能的。

**示例 2:**

输入: 2, [[1,0],[0,1]]

输出: false

解释: 总共有 2 门课程。学习课程 1 之前，你需要先完成​课程 0；并且学习课程 0 之前，你还应先完成课程 1。这是不可能的。

**提示：**

输入的先决条件是由 边缘列表 表示的图形，而不是 邻接矩阵 。详情请参见图的表示法。

你可以假定输入的先决条件中没有重复的边。

1 <= numCourses <= 10^5

### 2 基础知识

广度优先搜索，拓扑排序（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题和LeetCode 210 题一模一样，只是把返回数组改成true或者false。

本题目的解法就是利用拓扑排序再加上广度优先搜索。拓扑排序的目的是求出每个节点的入度，入度为0肯定先学，然后再利用广度优先搜索，将入度为0的统计出来个数，再将与入度为0节点相邻节点的入度减一，最后比较节点数目就可以了。

本人代码：

|  |
| --- |
| class Solution {  public:      bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {          // 存储有向图          vector<vector<int>> edges;          // 存储每个节点的入度          vector<int> indeg;          edges.resize(numCourses);          indeg.resize(numCourses);          int i;          for(i = 0; i < prerequisites.size(); i++) {//构建有向图并存储每个结点的入度值              edges[prerequisites[i][1]].push\_back(prerequisites[i][0]);              indeg[prerequisites[i][0]]++;          }          queue<int> q;//队列存储入度为0的点（即为正确的学习顺序）          for(i = 0; i < indeg.size(); i++) {              if(indeg[i] == 0)//入度为0的点先存进队列中                  q.push(i);          }          int num = 0;//记录可以学习的课程数量          while(!q.empty()) {              int u = q.front();              q.pop();              num++;              for(i = 0; i < edges[u].size(); i++) {//u点为去除掉的点，所以要将从u点来的入度减1                  indeg[edges[u][i]]--;                  if(indeg[edges[u][i]] == 0)                      q.push(edges[u][i]);              }          }          if (num != numCourses) {//判断可以学习的结点数是否等于已知结点数              return false;          }          return true;      }  }; |

## [210. 课程表 II](https://leetcode-cn.com/problems/course-schedule-ii/)

### 1 原题目

现在你总共有 n 门课需要选，记为 0 到 n-1。

在选修某些课程之前需要一些先修课程。 例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们: [0,1]

给定课程总量以及它们的先决条件，返回你为了学完所有课程所安排的学习顺序。

可能会有多个正确的顺序，你只要返回一种就可以了。如果不可能完成所有课程，返回一个空数组。

**示例 1:**

输入: 2, [[1,0]]

输出: [0,1]

解释: 总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1] 。

**示例 2:**

输入: 4, [[1,0],[2,0],[3,1],[3,2]]

输出: [0,1,2,3] or [0,2,1,3]

解释: 总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。并且课程 1 和课程 2 都应该排在课程 0 之后。

  因此，一个正确的课程顺序是 [0,1,2,3] 。另一个正确的排序是 [0,2,1,3] 。

**说明:**

输入的先决条件是由边缘列表表示的图形，而不是邻接矩阵。详情请参见图的表示法。

你可以假定输入的先决条件中没有重复的边。

**提示:**

这个问题相当于查找一个循环是否存在于有向图中。如果存在循环，则不存在拓扑排序，因此不可能选取所有课程进行学习。

通过 DFS 进行拓扑排序 - 一个关于Coursera的精彩视频教程（21分钟），介绍拓扑排序的基本概念。

拓扑排序也可以通过 BFS 完成。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

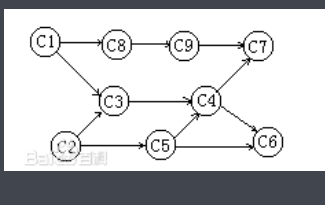
广度优先搜索，拓扑排序（解法一用到）

深度优先搜索（解法二用到）

### 3 解法

#### 3.1 解法1（C++已过）

拓扑排序百度百科：<https://baike.baidu.com/item/%E6%8B%93%E6%89%91%E6%8E%92%E5%BA%8F/5223807?fr=aladdin>



本题就是安排上课的顺序，例如上图，想要上C8这门课就要先上C1这门课，想要上C3这门课就要先上C1和C2这两两门课。拓扑排序也就是将这些节点排序成正确的顺序。

所以我们可以利用广度优先搜索来做，并且利用到图的入度这个知识，入度简单理解就是有几个箭头指向此结点。入度为0的点肯定是可以先学的课程，并且学完了那么从此门课指向其他课的入度减一，我们就要将入度为0的结点存到结果中，并且每存一个就要更新其他的结点的入度值，以此类推，最后要判断是否所有结点都在结果中（如果有的不在，就证明有向图中有环，不能完成拓扑排序）

本人代码：

|  |
| --- |
| class Solution {  public:      // 存储有向图      vector<vector<int>> edges;      // 存储每个节点的入度      vector<int> indeg;      // 存储答案      vector<int> result;      vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {          edges.resize(numCourses);          indeg.resize(numCourses);          int i;          for(i = 0; i < prerequisites.size(); i++) {//构建有向图并存储每个结点的入度值              edges[prerequisites[i][1]].push\_back(prerequisites[i][0]);              indeg[prerequisites[i][0]]++;          }          queue<int> q;//队列存储入度为0的点（即为正确的学习顺序）          for(i = 0; i < indeg.size(); i++) {              if(indeg[i] == 0)//入度为0的点先存进队列中                  q.push(i);          }          while(!q.empty()) {              int u = q.front();              q.pop();              result.push\_back(u);              for(i = 0; i < edges[u].size(); i++) {//u点为去除掉的点，所以要将从u点来的入度减1                  indeg[edges[u][i]]--;                  if(indeg[edges[u][i]] == 0)                      q.push(edges[u][i]);              }          }          if (result.size() != numCourses) {//判断结果中的结点数是否等于已知结点数              return {};          }          return result;      }  }; |

#### 3.2 解法2（C++已过）

此解法是利用深度优先搜索，个人感觉没有广度优先搜索好理解（因人而异喽）。此种解法建图是一样的，另外的就是要用数组记录每个结点的状态，0=未搜索，1=搜索中，2=已完成。

已完成搜索就是代表此节点可以存入结果中，搜索到0状态就递归进行深搜，搜索到1状态代表出现了环（无法拓扑排序），由于是深搜并且图的指向关系是先学指向后学，所以最后的结果要倒着输出（广搜解法是直接输出就可以，所以觉得好理解点），当然这是深搜广搜的差异带来的。

本人代码：

|  |
| --- |
| class Solution {  private:      // 存储有向图      vector<vector<int>> edges;      // 标记每个节点的状态：0=未搜索，1=搜索中，2=已完成      vector<int> visited;      // 用数组来模拟栈，下标0为栈底      vector<int> result;      // 判断有向图中是否有环      bool invalid;  public:      void dfs(int u) {          visited[u] = 1;//将结点标记为搜索中          int i;          for(i = 0; i < edges[u].size(); i++) {              //如果未搜索那么搜索相邻节点              if(visited[edges[u][i]] == 0) {                  dfs(edges[u][i]);              }              //如果找到了另外的结点在搜索中就说明找到了环              else if(visited[edges[u][i]] == 1) {                  invalid = true;                  return ;              }          }          visited[u] = 2;//将节点标记为已完成          result.push\_back(u);//压进栈中      }      vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {          edges.resize(numCourses);          visited.resize(numCourses);          int i;          for(i = 0; i < prerequisites.size(); i++) {//构建有向图并存储每个结点的入度值              edges[prerequisites[i][1]].push\_back(prerequisites[i][0]);          }          //每次挑选一个未搜索的节点，开始进行深度优先搜索          for (int i = 0; i < numCourses && !invalid; i++) {              if (visited[i] == 0) {                  dfs(i);              }          }          if (invalid) {              return {};          }          reverse(result.begin(), result.end());          return result;      }  }; |

## [126. 单词接龙 II](https://leetcode-cn.com/problems/word-ladder-ii/)

### 1 原题目

给定两个单词（beginWord 和 endWord）和一个字典 wordList，找出所有从 beginWord 到 endWord 的最短转换序列。转换需遵循如下规则：

每次转换只能改变一个字母。

转换后得到的单词必须是字典中的单词。

**说明:**

如果不存在这样的转换序列，返回一个空列表。

所有单词具有相同的长度。

所有单词只由小写字母组成。

字典中不存在重复的单词。

你可以假设 beginWord 和 endWord 是非空的，且二者不相同。

**示例 1:**

输入:

beginWord = "hit",

endWord = "cog",

wordList = ["hot","dot","dog","lot","log","cog"]

输出:

[

["hit","hot","dot","dog","cog"],

  ["hit","hot","lot","log","cog"]

]

**示例 2:**

输入:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

输出: []

解释: endWord "cog" 不在字典中，所以不存在符合要求的转换序列。

### 2 基础知识

深度优先搜索（解法一用到）

广度优先搜索（解法一用到）

无向图（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用单词的编号（单词在wordList的位置就是它的编号）建图，建图原则就是1号和2号单词相差字母数量为1，那就在二者之间建立一个无向边。建好图之后我们可以利用BFS对每个单词的层次进行记录。然后利用回溯和DFS就可以找出所有单词接龙的最短路径。

本人代码：

|  |
| --- |
| class Solution {  public:      vector<vector<int>>edges;//存储点和点之间的关系      vector<vector<string>>res;//结果      vector<int>lay;//存储每个点的层次      vector<string>te;      int endid;//endWord的编号      int diff(string &s1,string &s2) {//判断两个字符串是否不同字母只有一个          int dif = 0;          for (int i = 0; i < s1.size(); i++) {              if(s1[i] != s2[i])                  dif++;              if(dif > 1)                  return dif;          }          return dif;      }      void dfs(int preid,vector<string> &wordList) {          if (preid == endid) {//到达endWord              res.push\_back(te);//存进结果中              return ;          }          for (int i = 0; i < edges[preid].size(); i++) {              if (lay[edges[preid][i]] == lay[preid] + 1) {//在下一层寻找单词                  te.push\_back(wordList[edges[preid][i]]);                  dfs(edges[preid][i],wordList);                  te.pop\_back();              }          }          return ;      }      vector<vector<string>> findLadders(string beginWord, string endWord, vector<string>& wordList) {          int i,j;          int beginid = -1;          endid = -1;          edges.resize(wordList.size() + 10);          //建图(单词在wordList的位置就是单词的编号)          for (i = 0; i < wordList.size(); i++) {              if (wordList[i] == endWord)                  endid = i;              if(wordList[i] == beginWord)                  beginid = i;              for (j = i + 1; j < wordList.size(); j++) {                  if (diff(wordList[i], wordList[j]) == 1) {                      edges[i].push\_back(j);                      edges[j].push\_back(i);                  }              }          }          if (endid == -1)//wordList中不含有endWord              return {};          if (beginid == -1) {//如果wordList中不含有beginWord              beginid = wordList.size();//编号              for(i = 0; i < wordList.size(); i++) {                  if(diff(beginWord,wordList[i]) == 1) {                      edges[i].push\_back(beginid);                      edges[beginid].push\_back(i);                  }              }              wordList.push\_back(beginWord);//存到wordList中          }          lay.resize(wordList.size() + 10,-1);          queue<int> q;          q.push(beginid);          lay[beginid] = 0;          while (!q.empty()) {//利用BFS存储每个点的层次              int temp = q.front();              q.pop();              for (i = 0; i < edges[temp].size(); i++) {                  if (lay[edges[temp][i]] == -1) {                      lay[edges[temp][i]] = lay[temp] + 1;                      q.push(edges[temp][i]);                  }              }          }          te.push\_back(beginWord);          //回溯          dfs(beginid,wordList);//深度优先搜索          return res;      }  }; |

## [200. 岛屿数量](https://leetcode-cn.com/problems/number-of-islands/)

### 1 原题目

给定一个由 '1'（陆地）和 '0'（水）组成的的二维网格，计算岛屿的数量。一个岛被水包围，并且它是通过水平方向或垂直方向上相邻的陆地连接而成的。你可以假设网格的四个边均被水包围。

**示例 1:**

输入:

11110

11010

11000

00000

输出: 1

**示例 2:**

输入:

11000

11000

00100

00011

输出: 3

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

深度优先搜索（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

通过深度优先搜索将不同的小岛染成不同的色，记录出小岛的数量，关于深度优先搜索的知识大家自行百度，本人也搞不太明白。

本人代码：

int book[5000][5000],n,m,a[5000][5000];//全局变量

void dfs(int x,int y,int color){//深度优先搜索

int k,tx,ty;

int next[4][2]={0,1,1,0,0,-1,-1,0};//方向数组，分别是向右，下，左，上

a[x][y]=color;//染色

for(k=0;k<=3;k++){

tx=x+next[k][0];//下一步的横坐标

ty=y+next[k][1];//下一步的纵坐标

if(tx<0||tx>n||ty<0||ty>m)//判断是否越界

continue;

if(a[tx][ty]==1&&book[tx][ty]==0){//判断是否是陆地

book[tx][ty]=1;//标记这个点已经走过

dfs(tx,ty,color);//尝试下一个点

}

}

}

int numIslands(char\*\* grid, int gridSize, int\* gridColSize){

if(gridSize==0)

return 0;

n=gridSize-1;//记录数组的行数

m=\*gridColSize-1;//记录数组的列数

int i,j,num=0;

for(i=0;i<gridSize;i++)

for(j=0;j<\*gridColSize;j++){

if(grid[i][j]=='1')

a[i][j]=1;//数组a就是将grid数组转换成整型存储

else

a[i][j]=0;

book[i][j]=0;//标记数组的初始化

}

for(i=0;i<gridSize;i++)

for(j=0;j<\*gridColSize;j++){

if(a[i][j]==1){//对每个陆地进行染色判断它属于哪个小岛

num--;//小岛染色的编号，每个小岛不同编号，所以减1

book[i][j]=1;

dfs(i,j,num);

}

}

return -num;//返回小岛数量

}

## [1254. 统计封闭岛屿的数目](https://leetcode-cn.com/problems/number-of-closed-islands/)

### 1 原题目

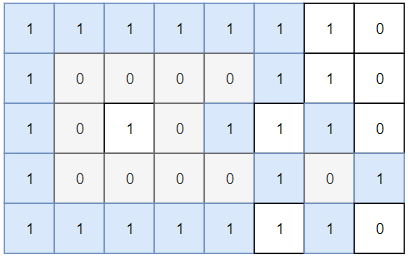
有一个二维矩阵 grid ，每个位置要么是陆地（记号为 0 ）要么是水域（记号为 1 ）。

我们从一块陆地出发，每次可以往上下左右 4 个方向相邻区域走，能走到的所有陆地区域，我们将其称为一座「岛屿」。

如果一座岛屿 完全 由水域包围，即陆地边缘上下左右所有相邻区域都是水域，那么我们将其称为 「封闭岛屿」。

请返回封闭岛屿的数目。

**示例 1：**



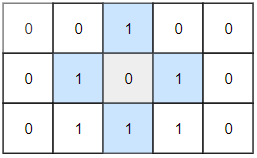
输入：grid = [[1,1,1,1,1,1,1,0],[1,0,0,0,0,1,1,0],[1,0,1,0,1,1,1,0],[1,0,0,0,0,1,0,1],[1,1,1,1,1,1,1,0]]

输出：2

解释：

灰色区域的岛屿是封闭岛屿，因为这座岛屿完全被水域包围（即被 1 区域包围）。

**示例 2：**



输入：grid = [[0,0,1,0,0],[0,1,0,1,0],[0,1,1,1,0]]

输出：1

**示例 3：**

输入：grid = [[1,1,1,1,1,1,1],

  [1,0,0,0,0,0,1],

  [1,0,1,1,1,0,1],

  [1,0,1,0,1,0,1],

  [1,0,1,1,1,0,1],

  [1,0,0,0,0,0,1],

[1,1,1,1,1,1,1]]

输出：2

**提示：**

1 <= grid.length, grid[0].length <= 100

0 <= grid[i][j] <=1

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

深度优先搜索（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

借助一个标记数组book，首先遍历四个边找到陆地进行递归深度优先搜索标记（从四个边连接起来的陆地不属于封闭陆地），然后遍历去除四条边后的地图，找到陆地且未被标记进行递归深度优先搜索标记。

本人代码：

int book[105][105],row,col;//book是标记数组

int move[4][2] = {-1,0,1,0,0,-1,0,1};//下一步的走法（上下左右）

void dfs(int p,int q,int \*\*grid){//深度优先搜索

    int np,nq,k;

    for(k = 0; k < 4; k++){//for循环对应的下一步到达的位置

        np = p + move[k][0];

        nq = q + move[k][1];

        if(np < 0 || nq < 0 || np >= row || nq >= col)//是否越界

            continue;

        if(grid[np][nq] == 0 && book[np][nq] == 0) {//如果下一步的位置是陆地且没被标记

            book[np][nq] = 1;//标记此位置

            dfs(np,nq,grid);//递归深度优先搜索

        }

    }

}

int closedIsland(int\*\* grid, int gridSize, int\* gridColSize){

    memset(book,0,sizeof(book));//标记数组初始化置为0

    row = gridSize;//记录grid数组的行数

    col = \*gridColSize;//记录grid数组的列数

    int i,j;

    //下面两个for循环的作用是将连接起来到四个边的陆地进行标记（不符合封闭岛屿的条件）

    for(i = 0; i < gridSize; i++) {

        if(grid[i][0] == 0 && book[i][0] == 0) {//是陆地且没被标记

            book[i][0] = 1;//标记此位置

            dfs(i,0,grid);//深度优先搜索

        }

        if(grid[i][\*gridColSize - 1] == 0 && book[i][\*gridColSize - 1] == 0) {

            book[i][\*gridColSize - 1] = 1;

            dfs(i,\*gridColSize - 1,grid);

        }

    }

    for(j = 0; j < \*gridColSize; j++) {

        if(grid[0][j] == 0 && book[0][j] == 0) {

            book[0][j] = 1;

            dfs(0,j,grid);

        }

        if(grid[gridSize - 1][j] == 0 && book[gridSize - 1][j] == 0) {

            book[gridSize - 1][j] = 1;

            dfs(gridSize - 1,j,grid);

        }

    }

    int num = 0;//num记录封闭岛屿的数量

    for(i = 1; i < gridSize - 1; i++) {//遍历去除四个边的grid数组

        for(j = 1; j < \*gridColSize - 1; j++) {

            if(grid[i][j] == 0 && book[i][j] == 0) {//是陆地且没被标记

                num++;//岛屿数量加1

                book[i][j] = 1;//标记此位置

                dfs(i,j,grid);//深度优先搜索

            }

        }

    }

    return num;

}

## [130. 被围绕的区域](https://leetcode-cn.com/problems/surrounded-regions/)

### 1 原题目

给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

**示例:**

X X X X

X O O X

X X O X

X O X X

运行你的函数后，矩阵变为：

X X X X

X X X X

X X X X

X O X X

**解释:**

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。 任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

深度优先搜索（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

因为题意说边界的O不会被替换掉，所以我们可以利用这点将边界的这四条边开始遍历一遍，通过深搜找出所有与边界连通的O的区域标记成b（这些是不用替换掉的），最后在

遍历一遍数组，将所有的b还原成O，O变成X。

本人代码：

int n,m;

void dfs(char \*\*board,int row,int col) {//深度优先搜索

board[row][col] = 'b';//将与边界连通的标记成b字母

int k,tx,ty;

int next[4][2]={0,1,1,0,0,-1,-1,0};//方向数组，分别是向右，下，左，上

for(k=0;k<=3;k++){

tx=row+next[k][0];//下一步的横坐标

ty=col+next[k][1];//下一步的纵坐标

if(tx<0||tx>n||ty<0||ty>m)//判断是否越界

continue;

if(board[tx][ty] == 'O'){

dfs(board,tx,ty);//尝试下一个点

}

}

return ;

}

void solve(char\*\* board, int boardSize, int\* boardColSize){

if(boardSize == 0)

return ;

int i,j;

n = boardSize-1;//记录数组的行数

m = \*boardColSize-1;//记录数组的列数

for(j=0;j<\*boardColSize;j++) {//找出上下边界中的O并进行深搜，与边界连通的O就不能被X填充

if(board[0][j] == 'O')

dfs(board,0,j);

if(board[boardSize-1][j] == 'O')

dfs(board,boardSize-1,j);

}

for(i=1;i<boardSize-1;i++) {//找出左右边界中的O并进行深搜

if(board[i][0] == 'O')

dfs(board,i,0);

if(board[i][\*boardColSize-1] == 'O')

dfs(board,i,\*boardColSize-1);

}

for(i=0;i<boardSize;i++)//将整个数组遍历一边

for(j=0;j<\*boardColSize;j++){

if(board[i][j] == 'O')

board[i][j] = 'X';

if(board[i][j] == 'b')

board[i][j] = 'O';

}

}

## [322. 零钱兑换](https://leetcode-cn.com/problems/coin-change/)

### 1 原题目

给定不同面额的硬币 coins 和一个总金额 amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 -1。

**示例 1:**

输入: coins = [1, 2, 5], amount = 11

输出: 3

解释: 11 = 5 + 5 + 1

**示例 2:**

输入: coins = [2], amount = 3

输出: -1

**说明:**

你可以认为每种硬币的数量是无限的。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

深度优先搜索（解法一用到）

动态规划（解法二用到）

### 3 解法

#### 3.1 解法1（C++已过）

这道题就是深度优先搜索加剪枝，一开始自己写dfs函数的for循环写的不好导致超时，一开始本人做的时候剪枝和贪心（其实就是先用面值大的硬币）都想到了，就是for循环超时。

后来看到其他人直接用剩余钱数除以当前数组中的钱数来直接确定钱数的上界，恍然大悟啊！（自己之前写的是一个个加，属实太慢）

本人代码：

class Solution {

    int min = INT\_MAX;//用来记录最少硬币数

public:

    void dfs(vector<int>& coins, long long am,int index,int num) {//am是当前剩下的钱，index是数组下标，num是当前已用的硬币数

        if(am == 0) {//找到一组解使得钱凑够了

            min = min > num ? num : min;//通过比较得到硬币最少的解

            return ;

        }

        if(index == -1)//下标-1就要返回了

            return ;

        for(int k = am / coins[index]; k >= 0; k--) {//直接算剩下的钱能用几张当前数组的钱凑然后遍历

            if(num + k >= min)//如果当前的硬币数加上接下来用的硬币数超过已凑够的某种硬币数

                return ;//直接返回

            dfs(coins,am - k \* coins[index],index - 1,num + k);

        }

        return ;

    }

    int coinChange(vector<int>& coins, int amount) {

        sort(coins.begin(),coins.end());//将硬币按照面值从小到大排序

        dfs(coins,amount,coins.size() - 1,0);//深度优先搜索加剪枝

        if(min != INT\_MAX)

            return min;

        return -1;

    }

};

#### 3.2 解法2（C++已过）

利用动态规划来求解，dp[i]表示凑成i元钱用的最少硬币数。状态方程的推导是动态规划的最主要的东西，可以说状态方程解出来那这道题就解决了。

这道题的状态方程的推导可以参考官方题解：<https://leetcode-cn.com/problems/coin-change/solution/322-ling-qian-dui-huan-by-leetcode-solution/>

我实在是说不明白状态方程的推导，现在懂了，我怕以后再看这道题就蒙了。

本人代码：

class Solution {

public:

    int coinChange(vector<int>& coins, int amount) {

        //dp[i]表示凑成金额为i的钱需要的最少硬币数

        vector<int>dp(amount + 1, amount + 1);//开辟大小为amount+1的数组并赋初值为amount+1

        dp[0] = 0;

        for(int i = 1; i <= amount; i++) {//表示当前要凑得金额

            for(int j = 0; j < coins.size(); j++) {//已知硬币的金额

                if(coins[j] <= i)//2块钱总不能用5块来凑吧，所以有这个条件

                    dp[i] = min(dp[i],dp[i - coins[j]] + 1);//状态方程

            }

        }

        //凑成10块钱最多使用10个一块钱硬币，所以如果大于本身了就要返回-1，代表凑不出

        return dp[amount] > amount ? -1 : dp[amount];

    }

};

## [329. 矩阵中的最长递增路径](https://leetcode-cn.com/problems/longest-increasing-path-in-a-matrix/)

### 1 原题目

给定一个整数矩阵，找出最长递增路径的长度。

对于每个单元格，你可以往上，下，左，右四个方向移动。 你不能在对角线方向上移动或移动到边界外（即不允许环绕）。

**示例 1:**

输入: nums =

[

[9,9,4],

[6,6,8],

[2,1,1]

]

输出: 4

解释: 最长递增路径为 [1, 2, 6, 9]。

**示例 2:**

输入: nums =

[

[3,4,5],

[3,2,6],

[2,2,1]

]

输出: 4

解释: 最长递增路径是 [3, 4, 5, 6]。注意不允许在对角线方向上移动。

### 2 基础知识

记忆化深度优先搜索（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用记忆化搜索可以提高代码的运行效率，因为在进行深度优先搜索的时候可以直接借用之前运算出来的结果，而不是每次都进行重复的计算，大大的提高了效率。

本人代码：

|  |
| --- |
| class Solution {  public:      int cool[4][2] = {0,1,0,-1,-1,0,1,0};//上下左右      int row,col;      //记忆化深度优先搜索      int dfs(int r,int c,vector<vector<int>>& matrix,vector<vector<int>>& res) {          for(int i = 0; i < 4; i++) {//只能上下左右移动              int x = r + cool[i][0];              int y = c + cool[i][1];              if(x < 0 || y < 0 || x >= row || y >= col)//越界                  continue;              if(matrix[x][y] <= matrix[r][c])//寻找的是递增路径                  continue;              if(res[x][y] != 0) {//记忆化搜索，下一步不是0可以直接拿过来用                  res[r][c] = max(res[r][c],1 + res[x][y]);//取最大值              }              else {//是0的话还要继续深度优先搜索                  res[r][c] = max(res[r][c],1 + dfs(x,y,matrix,res));//取              }          }          if(res[r][c] == 0)//每个格子自身算一个长度              res[r][c] = 1;          return res[r][c];      }      int longestIncreasingPath(vector<vector<int>>& matrix) {          if (matrix.size() == 0 || matrix[0].size() == 0)              return 0;          row = matrix.size();          col = matrix[0].size();          int ans = 0;          //记录每个单元格的最长递增路径的长度          vector<vector<int>> res(row,vector<int>(col,0));          for(int i = 0; i < row; i++) {              for(int j = 0; j < col; j++) {                  dfs(i,j,matrix,res);                  ans = max(ans,res[i][j]);//找最大值              }          }          return ans;      }  }; |

## [面试题 08.11. 硬币](https://leetcode-cn.com/problems/coin-lcci/)

### 1 原题目

硬币。给定数量不限的硬币，币值为25分、10分、5分和1分，编写代码计算n分有几种表示法。(结果可能会很大，你需要将结果模上1000000007)

**示例1:**

输入: n = 5

输出：2

解释: 有两种方式可以凑成总金额:

5=5

5=1+1+1+1+1

**示例2:**

输入: n = 10

输出：4

解释: 有四种方式可以凑成总金额:

10=10

10=5+5

10=5+1+1+1+1+1

10=1+1+1+1+1+1+1+1+1+1

**说明：**

注意:

你可以假设：

0 <= n (总金额) <= 1000000

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

第一层的for循环可以看成每次是由哪些硬币组成钱数，一开始是只有25分组成，再一次就是由25，10这两个组成，再一次就是25，10，5组成，再一次就是25，10，5，1组成。（这样可以保证没有重复的）

第二层for循环的作用就是滚动的更新值。

参考链接：<https://leetcode-cn.com/problems/coin-lcci/solution/ying-bi-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      int dp[1000005];      int waysToChange(int n) {          memset(dp,0,sizeof(dp));          dp[0] = 1;          int i,j,arr[4] = {25,10,5,1};          for(i = 0; i < 4; i++) {              int temp = arr[i];              for(j = temp; j <= n; j++)                  dp[j] = (dp[j] + dp[j - temp]) % 1000000007;          }          return dp[n];      }  }; |

## [394. 字符串解码](https://leetcode-cn.com/problems/decode-string/)

### 1 原题目

给定一个经过编码的字符串，返回它解码后的字符串。

编码规则为: k[encoded\_string]，表示其中方括号内部的 encoded\_string 正好重复 k 次。注意 k 保证为正整数。

你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。

此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。

**示例:**

s = "3[a]2[bc]", 返回 "aaabcbc".

s = "3[a2[c]]", 返回 "accaccacc".

s = "2[abc]3[cd]ef", 返回 "abcabccdcdcdef".

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

这种解法比较暴力，就是将’[]’中的字母存起来，在将’[]’前的数字计算出来，然后进行存储，详细请见代码。

本人代码：

char \* decodeString(char \* s){

int len=strlen(s),i,l=0,j,k,temp,l1,z,flag;//l1和flag都是用来记录临时值的

char \*stack=(char \*)malloc(5000\*sizeof(char));//最后返回的数组

char arr[5000]={'0'};//辅助数组

memset(stack,0,sizeof(stack));//数组初始化

for(i=0;i<len;i++){

if(s[i]!=']'){

stack[l++]=s[i];

}

else{

k=0;

temp=0;//temp是保存数字是多少

l1=l;//临时保存此时l的位置，即此时'[]'中字母结束的位置

for(j=l-1;j>=0;j--)

if(stack[j]=='[')//找到'['的位置

break;

l=j-1;//'['的前一个即为数字的开始

for(j=l;j>=0;j--){

if(stack[j]>='0'&&stack[j]<='9')

temp=temp+(stack[j]-'0')\*pow(10,k++);//计算出数字是多少

else

break;

}

flag=j+1;//记录数字的起始位置，方便之后将数字覆盖存储

k=0;

for(j=l+2;j<l1;j++)//l+2即为'[]'中字母的起始位置，l1即为结束位置

arr[k++]=stack[j];//将'[]'里的字母存入辅助数组中

l=flag;

for(j=0;j<temp;j++)

for(z=0;z<k;z++)

stack[l++]=arr[z];//从数字的起始位置进行存储

}

}

stack[l]='\0';//最后数组的结束标志

return stack;//返回数组

}

## [121. 买卖股票的最佳时机](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/)

### 1 原题目

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

如果你最多只允许完成一笔交易（即买入和卖出一支股票），设计一个算法来计算你所能获取的最大利润。

注意你不能在买入股票前卖出股票。

**示例 1:**

输入: [7,1,5,3,6,4]

输出: 5

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。

注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格。

**示例 2:**

输入: [7,6,4,3,1]

输出: 0

解释: 在这种情况下, 没有交易完成, 所以最大利润为 0。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

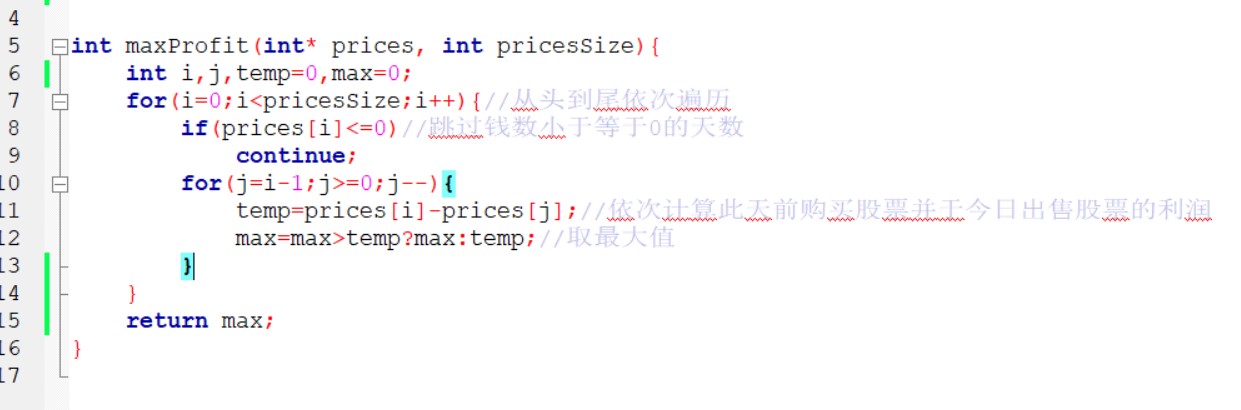
无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过将今日前购买股票并于今日出售的所有情况都计算一遍，之后每次比较取大的值，即为最大利润。

本人代码：

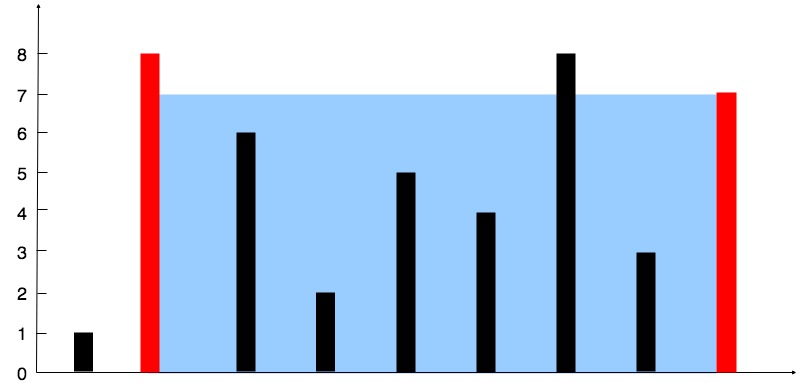


## [11. 盛最多水的容器](https://leetcode-cn.com/problems/container-with-most-water/)

### 1 原题目

给定 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0)。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

**说明：**你不能倾斜容器，且 n 的值至少为 2。



图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

**示例:**

输入: [1,8,6,2,5,4,8,3,7]

输出: 49

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

运用双指针，一个指头，一个指尾，而且容器的宽取决于短边，每计算一次后，指针移动的原则是短边的指针朝着长边的那一端移动。

本人代码：



## [42. 接雨水](https://leetcode-cn.com/problems/trapping-rain-water/)

### 1 原题目

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。



上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。 感谢 Marcos 贡献此图。

**示例:**

输入: [0,1,0,2,1,0,1,3,2,1,2,1]

输出: 6

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

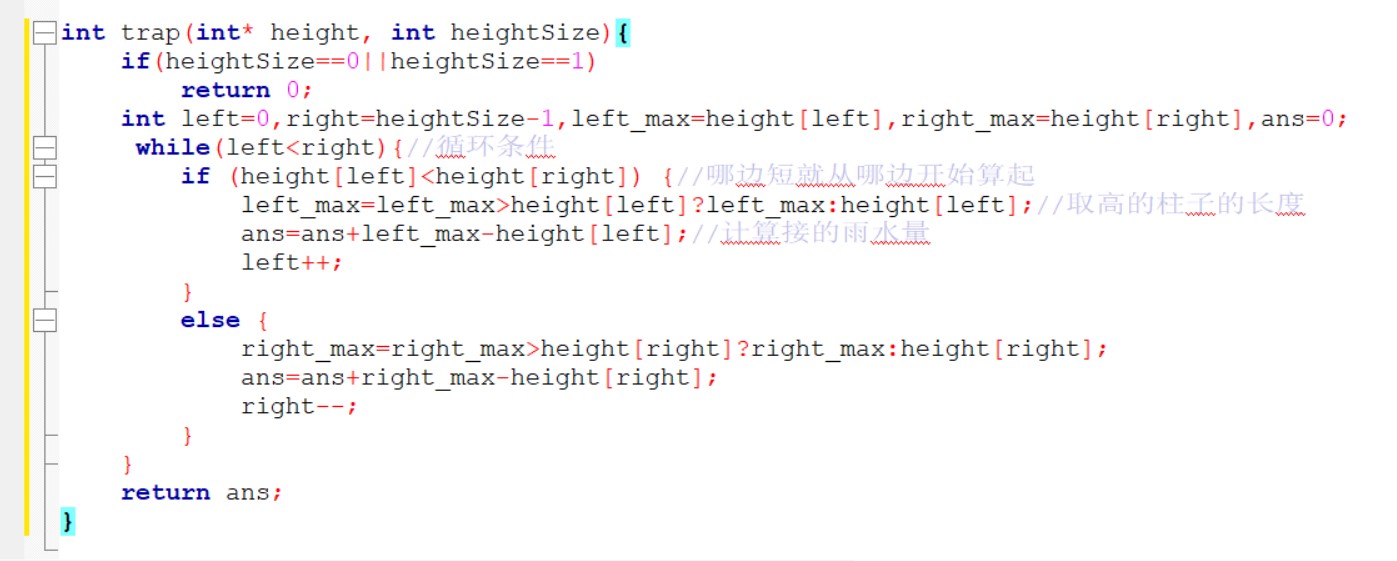
无。

### 3 解法

#### 3.1 解法1（C语言已过）

利用双指针，一个指针指头，一个指针指尾，比较两个指针所指数值的大小，从小的一端开始计算，比如左边小，就从左到右，右边小，就从右到左，雨水量的计算大家看代码理解吧，我表述不明白。

本人代码：



## [398. 随机数索引](https://leetcode-cn.com/problems/random-pick-index/)

### 1 原题目

给定一个可能含有重复元素的整数数组，要求随机输出给定的数字的索引。 您可以假设给定的数字一定存在于数组中。

**注意：**

数组大小可能非常大。 使用太多额外空间的解决方案将不会通过测试。

**示例:**

int[] nums = new int[] {1,2,3,3,3};

Solution solution = new Solution(nums);

// pick(3) 应该返回索引 2,3 或者 4。每个索引的返回概率应该相等。

solution.pick(3);

// pick(1) 应该返回 0。因为只有nums[0]等于1。

solution.pick(1);

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

蓄水池抽样算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

这道题其他部分都不难，难就难在那随机取索引的三两行代码，这就要用到蓄水池抽样算法的思想。

**蓄水池抽样算法介绍：**

**自行百度，本人没找到很好的。**

本人代码：

typedef struct {//结构体

int \*arr;//数组

int len;//数组长度

} Solution;

Solution\* solutionCreate(int\* nums, int numsSize) {

Solution\* obj = (Solution\*)malloc(sizeof(Solution));

obj->arr = nums;

obj->len = numsSize;

return obj;

}

int solutionPick(Solution\* obj, int target) {

int i=0,ans,num=0;//num用来记录与所给值相等数字的数量

for(i=0; i<obj->len; i++){

if(obj->arr[i] == target){//if开始的三行是核心代码，也是最重要的

num++;

ans=(rand()%num==0)?i:ans;//随机取样

}

}

return ans;

}

void solutionFree(Solution\* obj) {

free(obj);//释放空间

}

/\*\*

\* Your Solution struct will be instantiated and called as such:

\* Solution\* obj = solutionCreate(nums, numsSize);

\* int param\_1 = solutionPick(obj, target);

\* solutionFree(obj);

\*/

## [382. 链表随机节点](https://leetcode-cn.com/problems/linked-list-random-node/)

### 1 原题目

给定一个单链表，随机选择链表的一个节点，并返回相应的节点值。保证每个节点被选的概率一样。

**进阶:**

如果链表十分大且长度未知，如何解决这个问题？你能否使用常数级空间复杂度实现？

**示例:**

// 初始化一个单链表 [1,2,3].

ListNode head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(3);

Solution solution = new Solution(head);

// getRandom()方法应随机返回1,2,3中的一个，保证每个元素被返回的概率相等。

solution.getRandom();

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

蓄水池抽样算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

这道题其他部分都不难，难就难在那随机返回节点值的三两行代码，这就要用到蓄水池抽样算法的思想。

**蓄水池抽样算法介绍：**

**自行百度，本人没找到很好的。**

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

typedef struct {

int arr[50000];

int num;

} Solution;

/\*\* @param head The linked list's head.

Note that the head is guaranteed to be not null, so it contains at least one node. \*/

Solution\* solutionCreate(struct ListNode\* head) {

Solution \*obj=(Solution \*)malloc(sizeof(Solution));

int i=0;

while(head!=NULL){

obj->arr[i++]=head->val;//将head的节点值存到obj中的数组中

head=head->next;

}

obj->num=i;//num是记录节点的总数量

return obj;

}

/\*\* Returns a random node's value. \*/

int solutionGetRandom(Solution\* obj) {

int i,res=0;

for(i=0;i<obj->num;i++){//从此处开始的两行是核心代码

res=(rand()%(i+1)==0)?obj->arr[i]:res;//随机等概率取节点值

}

return res;

}

void solutionFree(Solution\* obj) {

free(obj);//释放

}

/\*\*

\* Your Solution struct will be instantiated and called as such:

\* Solution\* obj = solutionCreate(head);

\* int param\_1 = solutionGetRandom(obj);

\* solutionFree(obj);

\*/

## [380. 常数时间插入、删除和获取随机元素](https://leetcode-cn.com/problems/insert-delete-getrandom-o1/)

### 1 原题目

设计一个支持在平均 时间复杂度 O(1) 下，执行以下操作的数据结构。

insert(val)：当元素 val 不存在时，向集合中插入该项。

remove(val)：元素 val 存在时，从集合中移除该项。

getRandom：随机返回现有集合中的一项。每个元素应该有相同的概率被返回。

**示例 :**

// 初始化一个空的集合。

RandomizedSet randomSet = new RandomizedSet();

// 向集合中插入 1 。返回 true 表示 1 被成功地插入。

randomSet.insert(1);

// 返回 false ，表示集合中不存在 2 。

randomSet.remove(2);

// 向集合中插入 2 。返回 true 。集合现在包含 [1,2] 。

randomSet.insert(2);

// getRandom 应随机返回 1 或 2 。

randomSet.getRandom();

// 从集合中移除 1 ，返回 true 。集合现在包含 [2] 。

randomSet.remove(1);

// 2 已在集合中，所以返回 false 。

randomSet.insert(2);

// 由于 2 是集合中唯一的数字，getRandom 总是返回 2 。

randomSet.getRandom();

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

蓄水池抽样算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

因为本解法时间复杂度不符合要求，所以删除和插入元素好办，至于随机获取元素则需要用到蓄水池抽样算法的思想（本解法中我的代码貌似时间复杂度不符合要求）

**蓄水池抽样算法介绍：**

**自行百度，本人没找到很好的。**

本人代码：

typedef struct {

int arr[50000];

int num;//记录数组中已有多少个元素

} RandomizedSet;

/\*\* Initialize your data structure here. \*/

RandomizedSet\* randomizedSetCreate() {

RandomizedSet \*obj=(RandomizedSet \*)malloc(sizeof(RandomizedSet));

obj->num=0;

memset(obj->arr,0,sizeof(obj->arr));//数组的初始化

return obj;

}

/\*\* Inserts a value to the set. Returns true if the set did not already contain the specified element. \*/

bool randomizedSetInsert(RandomizedSet\* obj, int val) {

int i,j;

if(obj->num==0){//数组为空时，直接插入就可以

obj->arr[0]=val;

obj->num=1;

return true;

}

else{//我的数组中元素的存储是按从小到大顺序存储的

for(i=0;i<obj->num;i++){

if(obj->arr[i]==val)

break;

if(obj->arr[obj->num-1]<val){

obj->arr[obj->num]=val;

obj->num++;

return true;

}

if(obj->arr[i]>val){

for(j=obj->num;j>i;j--)

obj->arr[j]=obj->arr[j-1];//元素后移

obj->arr[i]=val;//插入新元素

obj->num++;

return true;

}

}

}

return false;

}

/\*\* Removes a value from the set. Returns true if the set contained the specified element. \*/

bool randomizedSetRemove(RandomizedSet\* obj, int val) {

int i,j;

if(obj->num==0)

return false;

for(i=0;i<obj->num;i++){

if(obj->arr[i]>val)

break;

if(obj->arr[i]==val){

for(j=i;j<obj->num-1;j++)

obj->arr[j]=obj->arr[j+1];//元素前移

obj->num--;

return true;

}

}

return false;

}

/\*\* Get a random element from the set. \*/

int randomizedSetGetRandom(RandomizedSet\* obj) {

int res=0,i;

for(i=0;i<obj->num;i++){//这两行是随机取的核心代码

res=(rand()%(i+1)==0)?obj->arr[i]:res;

}

return res;

}

void randomizedSetFree(RandomizedSet\* obj) {

free(obj);//释放

}

/\*\*

\* Your RandomizedSet struct will be instantiated and called as such:

\* RandomizedSet\* obj = randomizedSetCreate();

\* bool param\_1 = randomizedSetInsert(obj, val);

\* bool param\_2 = randomizedSetRemove(obj, val);

\* int param\_3 = randomizedSetGetRandom(obj);

\* randomizedSetFree(obj);

\*/

## [73. 矩阵置零](https://leetcode-cn.com/problems/set-matrix-zeroes/)

### 1 原题目

给定一个 m x n 的矩阵，如果一个元素为 0，则将其所在行和列的所有元素都设为 0。请使用原地算法。

**示例 1:**

输入:

[

  [1,1,1],

  [1,0,1],

  [1,1,1]

]

输出:

[

  [1,0,1],

  [0,0,0],

  [1,0,1]

]

**示例 2:**

输入:

[

  [0,1,2,0],

  [3,4,5,2],

  [1,3,1,5]

]

输出:

[

  [0,0,0,0],

  [0,4,5,0],

  [0,3,1,0]

]

**进阶:**

一个直接的解决方案是使用  O(mn) 的额外空间，但这并不是一个好的解决方案。

一个简单的改进方案是使用 O(m + n) 的额外空间，但这仍然不是最好的解决方案。

你能想出一个常数空间的解决方案吗？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将二维数组遍历一遍，如果出现了0元素，就将这一行和这一列的首元素置为零，由于第0行和第0列的首元素是重合的，所以我们用flag来标记第0列是否置为0。然后在遍历数组中除了第0行和第0列首元素的其他行和列的首元素，如果是0就将这一行或列的元素全置为0，最后在判断第0行和第0列是否全置为0。

本人代码：

void setZeroes(int\*\* matrix, int matrixSize, int\* matrixColSize){

int i,j,flag=0;//flag用来标记第0列是否需要置为0

for(i=0;i<matrixSize;i++){//遍历一遍数组

for(j=0;j<\*matrixColSize;j++){

if(matrix[i][j]==0){//将出现0元素的那一行和那一列的首元素标记为0

if(j==0)

flag=1;//如果是第0列需要标记就用flag来标记

else

matrix[0][j]=0;

matrix[i][0]=0;

}

}

}

for(i=1;i<matrixSize;i++){//先判断除去第0行以外行的首元素

if(matrix[i][0]==0){//如果是零

for(j=0;j<\* matrixColSize;j++)//这一行全置为0

matrix[i][j]=0;

}

}

for(j=1;j<\* matrixColSize;j++){//先判断除去第0列以外列的首元素

if(matrix[0][j]==0){//如果是0

for(i=0;i<matrixSize;i++)//这一列全置为0

matrix[i][j]=0;

}

}

if(matrix[0][0]==0)//判断第0行的首元素是否为0

for(j=0;j<\* matrixColSize;j++)

matrix[0][j]=0;

if(flag==1)//判断第0列的首元素是否被标记

for(i=0;i<matrixSize;i++)

matrix[i][0]=0;

}

## [74. 搜索二维矩阵](https://leetcode-cn.com/problems/search-a-2d-matrix/)

### 1 原题目

编写一个高效的算法来判断 m x n 矩阵中，是否存在一个目标值。该矩阵具有如下特性：

每行中的整数从左到右按升序排列。

每行的第一个整数大于前一行的最后一个整数。

**示例 1:**

输入:

matrix = [

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

target = 3

输出: true

**示例 2:**

输入:

matrix = [

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

target = 13

输出: false

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

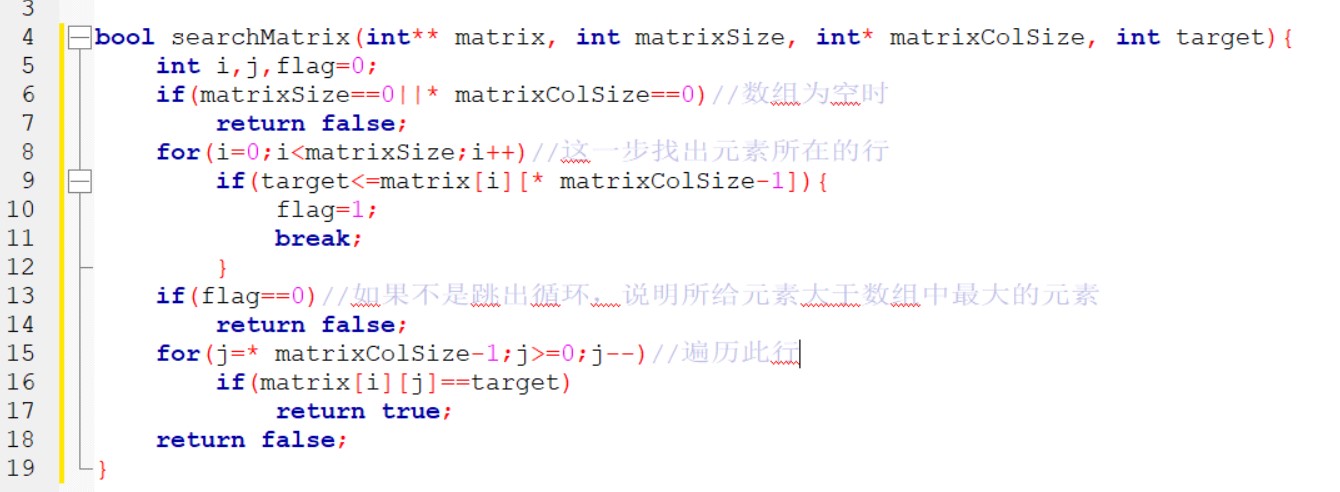
无。

### 3 解法

#### 3.1 解法1（C语言已过）

因为题意告诉我们每一行的首元素都大于前一行的尾元素，而每一行都是升序排列，所以我们先通过将元素和每一行的尾元素进行比较判断出所找的元素应该在哪一行，然后再遍历此行元素判断是否等于所给元素。

本人代码：



## [240. 搜索二维矩阵 II](https://leetcode-cn.com/problems/search-a-2d-matrix-ii/)

### 1 原题目

编写一个高效的算法来搜索 m x n 矩阵 matrix 中的一个目标值 target。该矩阵具有以下特性：

每行的元素从左到右升序排列。

每列的元素从上到下升序排列。

**示例:**

现有矩阵 matrix 如下：

[

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

]

给定 target = 5，返回 true。

给定 target = 20，返回 false。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

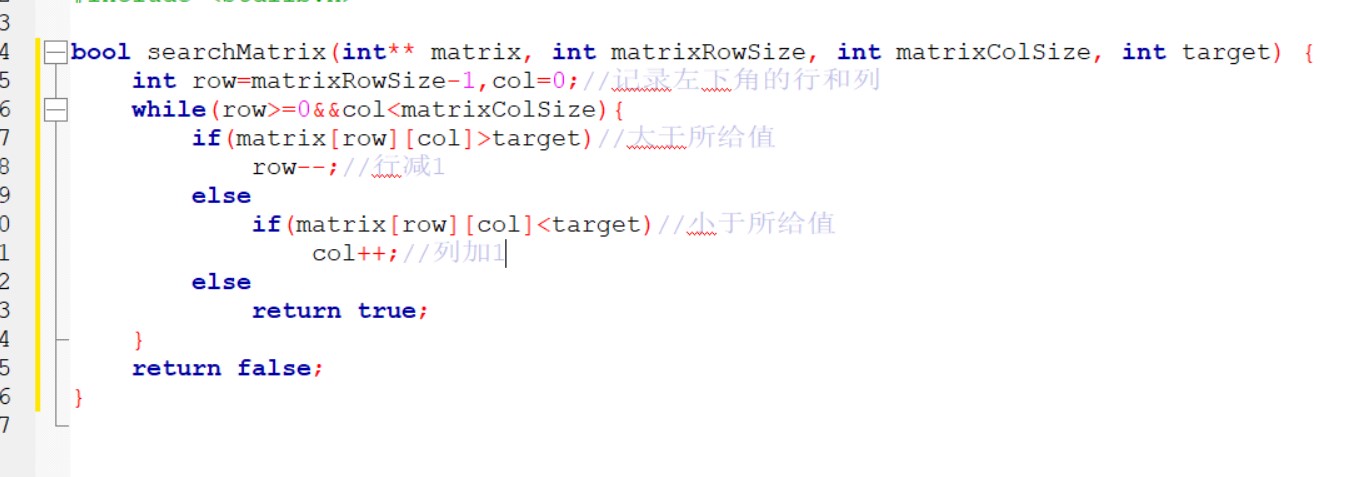
无。

### 3 解法

#### 3.1 解法1（C语言已过）

此二维数组是从左到右升序排列，从上到下升序排列，我们可以发现如果从数组的左下角开始搜索，正好是如果给定值大于数组当前值，就将列数加1，给定值小于数组当前值，就将行数减1，这样就可以完成搜索了。

本人代码：



## [48. 旋转图像](https://leetcode-cn.com/problems/rotate-image/)

### 1 原题目

给定一个 n × n 的二维矩阵表示一个图像。

将图像顺时针旋转 90 度。

**说明：**

你必须在原地旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。

**示例 1:**

给定 matrix =

[

[1,2,3],

[4,5,6],

[7,8,9]

],

原地旋转输入矩阵，使其变为:

[

[7,4,1],

[8,5,2],

[9,6,3]

]

**示例 2:**

给定 matrix =

[

[ 5, 1, 9,11],

[ 2, 4, 8,10],

[13, 3, 6, 7],

[15,14,12,16]

],

原地旋转输入矩阵，使其变为:

[

[15,13, 2, 5],

[14, 3, 4, 1],

[12, 6, 8, 9],

[16, 7,10,11]

]

### 2 基础知识

#### 2.1数据结构

无。

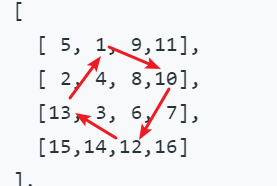
#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

找出规律即可，不过不太好找。元素的旋转是按照下图的样式旋转的。



其实规律就是(i,j) , (j, n-i-1), (n-i-1, n-j-1), (n-j-1, i)这四个索引号上的数交换.

本人代码：

void rotate(int\*\* matrix, int matrixSize, int\* matrixColSize){

    int i,j,temp,n=matrixSize;

    //其实规律就是(i,j) , (j,n-i-1), (n-i-1,n-j-1), (n-j-1,i)这四个索引号上的数交换.

    for(i=0;i<n;i++){//交换元素

        for(j=i;j<n-i-1;j++){

            temp=matrix[i][j];

            matrix[i][j]=matrix[n-j-1][i];

            matrix[n-j-1][i]=matrix[n-i-1][n-j-1];

            matrix[n-i-1][n-j-1]=matrix[j][n-i-1];

            matrix[j][n-i-1]=temp;

        }

    }

}

#### 3.2 解法2（C++语言已过）

先将数组中的元素按照水平中线翻转，然后再按照主对角线进行翻转。

本人代码：

class Solution {

public:

    void rotate(vector<vector<int>>& matrix) {

        int n = matrix.size();

        //水平翻转

        for(int i = 0; i < n / 2; ++i) {

            for(int j = 0; j < n; ++j) {

                swap(matrix[i][j], matrix[n - i - 1][j]);

            }

        }

        //主对角线翻转

        for(int i = 0; i < n; ++i) {

            for(int j = 0; j < i; ++j) {

                swap(matrix[i][j], matrix[j][i]);

            }

        }

    }

};

## [54. 螺旋矩阵](https://leetcode-cn.com/problems/spiral-matrix/)

### 1 原题目

给定一个包含 m x n 个元素的矩阵（m 行, n 列），请按照顺时针螺旋顺序，返回矩阵中的所有元素。

**示例 1:**

输入:

[

[ 1, 2, 3 ],

[ 4, 5, 6 ],

[ 7, 8, 9 ]

]

输出: [1,2,3,6,9,8,7,4,5]

**示例 2:**

输入:

[

[1, 2, 3, 4],

[5, 6, 7, 8],

[9,10,11,12]

]

输出: [1,2,3,4,8,12,11,10,9,5,6,7]

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

设置上下左右四个边界，在进行顺时针螺旋存储时只要每一步不超过此步对应的边界就可以了。

本人代码：

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int\* spiralOrder(int\*\* matrix, int matrixSize, int\* matrixColSize, int\* returnSize){

if(matrixSize==0||\* matrixColSize==0){//数组为空时

\* returnSize=0;

int \*arr=(int \*)malloc(2\*sizeof(int));

return arr;

}

int \*arr=(int \*)malloc((matrixSize\*(\* matrixColSize)+1)\*sizeof(int)),i=0,j=0,k=0;

memset(arr,0,sizeof(arr));//一维数组的初始化

int up=-1,down=matrixSize,left=-1,right=\* matrixColSize;//设置上下左右的边界

while(k<matrixSize\*(\* matrixColSize)){

while(j<right&&k<matrixSize\*(\* matrixColSize))//向右

arr[k++]=matrix[i][j++];

j--;//上一步的j的自加会越界，所以减一

i++;//防止在拐角处的元素多存储一次

up++;//遍历完毕上面的一行，更改上边界

while(i<down&&k<matrixSize\*(\* matrixColSize))//向下

arr[k++]=matrix[i++][j];

i--;//上一步的i的自加会越界，所以减一

j--;//防止在拐角处的元素多存储一次

right--;//遍历完毕右面的一列，更改右边界

while(j>left&&k<matrixSize\*(\* matrixColSize))//向左

arr[k++]=matrix[i][j--];

j++;//上一步的j的自减会越界，所以加一

i--;//防止在拐角处的元素多存储一次

down--;//遍历完毕下面的一行，更改下边界

while(i>up&&k<matrixSize\*(\* matrixColSize))//向上

arr[k++]=matrix[i--][j];

i++;//上一步的i的自减会越界，所以加一

j++;//防止在拐角处的元素多存储一次

left++;//遍历完毕左面的一列，更改左边界

}

\*returnSize=matrixSize\*(\* matrixColSize);

return arr;

}

## [59. 螺旋矩阵 II](https://leetcode-cn.com/problems/spiral-matrix-ii/)

### 1 原题目

给定一个正整数 n，生成一个包含 1 到 n2 所有元素，且元素按顺时针顺序螺旋排列的正方形矩阵。

**示例:**

输入: 3

输出:

[

[ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ]

]

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

因为数字是从1到n\*n，所以二维数组就是一个正方形，也就是说我们不用考虑特殊情况，设置上下左右四个边界，在进行顺时针螺旋存储时只要每一步不超过此步对应的边界就可以了。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int\*\* generateMatrix(int n, int\* returnSize, int\*\* returnColumnSizes){

int \*\*arr=(int \*\*)malloc(sizeof(int \*)\*n);//开辟n行

int \*col=(int \*)malloc(sizeof(int)\*n);

int i,j,k=1;

for(i=0; i<n; i++){

arr[i]=(int\*)malloc(sizeof(int)\*n);//每一行开辟n列

col[i]=n;

for(j=0;j<n;j++){

arr[i][j]=0;//数组的初始化

}

}

int up=-1,down=n,left=-1,right=n;//上下左右边界

i=0;

j=0;

while(k<=n\*n){

while(j<right&&k<=n\*n)//向右

arr[i][j++]=k++;

j--;//上一步的j的自加会越界，所以减一

i++;//防止在拐角处多存储一次元素

up++;//遍历完毕上面的一行，更改上边界

while(i<down&&k<=n\*n)//向下

arr[i++][j]=k++;

i--;//上一步的i的自加会越界，所以减一

j--;//防止在拐角处多存储一次元素

right--;//遍历完毕右面的一列，更改右边界

while(j>left&&k<=n\*n)//向左

arr[i][j--]=k++;

j++;//上一步的j的自减会越界，所以加一

i--;//防止在拐角处多存储一次元素

down--;//遍历完毕下面的一行，更改下边界

while(i>up&&k<=n\*n)//向上

arr[i--][j]=k++;

i++;//上一步的i的自减会越界，所以加一

j++;//防止在拐角处多存储一次元素

left++;//遍历完毕左面的一列，更改左边界

}

\*returnSize=n;

\*returnColumnSizes=col;

return arr;

}

## [138. 复制带随机指针的链表](https://leetcode-cn.com/problems/copy-list-with-random-pointer/)

### 1 原题目

给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。

要求返回这个链表的深拷贝。

**示例：**

图片包含 物体, 天线

描述已自动生成

输入：

{"$id":"1","next":{"$id":"2","next":null,"random":{"$ref":"2"},"val":2},"random":{"$ref":"2"},"val":1}

解释：

节点 1 的值是 1，它的下一个指针和随机指针都指向节点 2 。

节点 2 的值是 2，它的下一个指针指向 null，随机指针指向它自己。

**提示：**

你必须返回给定头的拷贝作为对克隆列表的引用。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

通过指针的变换来达到建立出来和他所给链表一模一样的链表。随机指针random也是如此。

第一步是在原链表的每个节点后建立一个新节点，第二步处理随机指针，第三步将链表分开成两个独立的一模一样的链表。

指针的变化如果感觉理解困难，大家可以动手在纸上画出来，这样可以帮助理解。

本人代码：

class Solution {

public:

Node\* copyRandomList(Node\* head) {

if(!head)

return NULL;

//第一步，复制原始的节点，连接自身后面

Node\* p=head;

while(p){

Node \*q=(Node \*)malloc(sizeof(Node));

q->val=p->val;//给新建的节点赋值

q->next=p->next;

p->next=q;

p=q->next;

}

//第二步，将添加的节点random 链接到原始节点random的下一个位置

Node\* p1=head;

while(p1){

Node\* q=p1->next;

if(p1->random==NULL)

q->random=NULL;

else

q->random=p1->random->next;

p1=q->next;//处理下一个节点

}

//第三步，恢复原始链表，得到新链表

Node\* p2=head;

Node\* copy=head->next;

Node\* copy1=head->next;

while(p2->next->next!=NULL){

p2->next=copy->next;

p2=copy->next;

copy->next=p2->next;

copy=p2->next;

}

p2->next=NULL;

copy->next=NULL;

return copy1;

}

};

## [206. 反转链表](https://leetcode-cn.com/problems/reverse-linked-list/)

### 1 原题目

反转一个单链表。

**示例:**

输入: 1->2->3->4->5->NULL

输出: 5->4->3->2->1->NULL

**进阶:**

你可以迭代或递归地反转链表。你能否用两种方法解决这道题？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将所给的链表的节点值按顺序存入到一个数组中，然后再按照数组元素的倒序将元素构成一个链表。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* reverseList(struct ListNode\* head){

struct ListNode \*ans=(struct ListNode \*)malloc(sizeof(struct ListNode));//ans作为返回链表的首个节点

if(head==NULL){//当传进来的链表为空时

ans=NULL;

return ans;

}

struct ListNode \*obj=ans;//真正的链表连接的操作用obj来操作

int arr[5000]={0},n=0;

while(head!=NULL){

arr[n++]=head->val;//将链表元素存进数组中

head=head->next;

}

n--;

obj->val=arr[n--];

while(n>=0){//构建新的链表

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr[n--];

obj->next=temp;

obj=obj->next;

}

obj->next=NULL;

return ans;

}

#### 3.2 解法2（C语言已过）

利用指针的操作变换来实现链表的反转，具体请看代码和注释。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* reverseList(struct ListNode\* head){

    struct ListNode \*prev = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*curr = (struct ListNode \*)malloc(sizeof(struct ListNode));

    curr = head;//curr指针指向当前要进行操作的节点

    prev = NULL;//prev为curr前面的一个节点

    while(head != NULL) {//循环终止条件

        head = head -> next;

        curr -> next = prev;

        prev = curr;

        curr = head;

    }

    return prev;

}

## [92. 反转链表 II](https://leetcode-cn.com/problems/reverse-linked-list-ii/)

### 1 原题目

反转从位置 m 到 n 的链表。请使用一趟扫描完成反转。

**说明:**

1 ≤ m ≤ n ≤ 链表长度。

**示例:**

**输入: 1->2->3->4->5->NULL, m = 2, n = 4**

**输出: 1->4->3->2->5->NULL**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

用两个数组来辅助做这道题，一个数组存放区间里的节点值，一个数组存放区间外的节点值，剩下的就是将数组元素按照正确的顺序插进链表中

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* reverseBetween(struct ListNode\* head, int m, int n){

struct ListNode \*ans=(struct ListNode \*)malloc(sizeof(struct ListNode));//ans是返回链表的首个节点

struct ListNode \*obj=ans;//用obj来进行一系列的操作

int arr[5000]={0},arr1[5000]={0},i=0,j=0,k,num=1;//arr存放区间外的元素，arr1存放区间内的元素

while(head!=NULL&&num<m){

arr[i++]=head->val;

head=head->next;

num++;

}

while(head!=NULL&&num<=n){

arr1[j++]=head->val;

head=head->next;

num++;

}

while(head!=NULL){

arr[i++]=head->val;

head=head->next;

num++;

}

j--;

k=i--;

i=0;

if(m==1){//判断区间是否从1开始

obj->val=arr1[j--];

while(j>=0){

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr1[j--];

obj->next=temp;

obj=obj->next;

}

while(i<k){

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr[i++];

obj->next=temp;

obj=obj->next;

}

obj->next=NULL;

}

else{

obj->val=arr[i++];

while(i<m-1){

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr[i++];

obj->next=temp;

obj=obj->next;

}

while(j>=0){

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr1[j--];

obj->next=temp;

obj=obj->next;

}

while(i<k){

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr[i++];

obj->next=temp;

obj=obj->next;

}

obj->next=NULL;

}

return ans;

}

#### 3.2 解法2（C语言已过）

先在链表中找出要进行反转的区间，然后对此区间进行反转，然后将反转后的区间链表重新连接到原链表的位置上，具体请看代码和注释。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* swap(struct ListNode\* head,struct ListNode\* tail){//反转所给区间内的链表

    struct ListNode \*prev = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*curr = (struct ListNode \*)malloc(sizeof(struct ListNode));

    curr = head;//curr指针指向当前要进行操作的节点

    prev = NULL;//prev为curr前面的一个节点

    while(head != NULL) {//循环终止条件

        head = head -> next;

        curr -> next = prev;

        prev = curr;

        curr = head;

    }

    return prev;

}

struct ListNode\* reverseBetween(struct ListNode\* head, int m, int n){

    int m1 = 1;//m1记录链表遍历的位置

    struct ListNode \*h = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*h1 = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*t = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*t1 = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*head1 = (struct ListNode \*)malloc(sizeof(struct ListNode));

    head1 = head;

    while(head != NULL) {//遍历链表来找出所要反转的链表区间

        if(m1 + 1 == m)

            h1 = head;//记录下m位置前一个的节点

        if(m1 == m)

            h = head;

        if(m1 == n) {

            t = head;

            t1 = head -> next;//记录n位置后一个的节点

            t -> next = NULL;

            break;

        }

        head = head -> next;

        m1++;

    }

    struct ListNode \*temp = (struct ListNode \*)malloc(sizeof(struct ListNode));

    temp = swap(h,t);//反转m到n区间的链表

    h1 -> next = temp;//将反转后的链表重新连接到原链表的位置上

    h -> next = t1;

    if(m == 1)//判断是否从第一个节点开始反转链表

        head1 = temp;

    return head1;

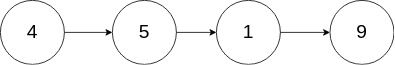
}

## [237. 删除链表中的节点](https://leetcode-cn.com/problems/delete-node-in-a-linked-list/)

### 1 原题目

请编写一个函数，使其可以删除某个链表中给定的（非末尾）节点，你将只被给定要求被删除的节点。

现有一个链表 -- head = [4,5,1,9]，它可以表示为:



**示例 1:**

输入: head = [4,5,1,9], node = 5

输出: [4,1,9]

解释: 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.

**示例 2:**

输入: head = [4,5,1,9], node = 1

输出: [4,5,9]

解释: 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

**说明:**

链表至少包含两个节点。

链表中所有节点的值都是唯一的。

给定的节点为非末尾节点并且一定是链表中的一个有效节点。

不要从你的函数中返回任何结果。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

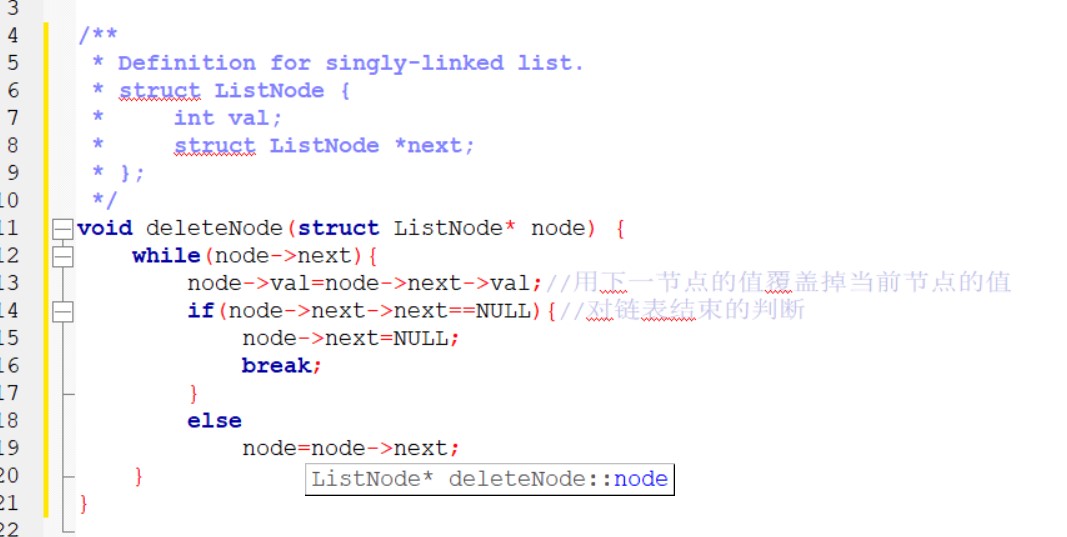
无。

### 3 解法

#### 3.1 解法1（C语言已过）

因为函数只给我们当前要删除的节点的指针，所以不能进行next指针跳过这个节点来达到删除的目的，但是我们可以像从数组中删除一个元素那样进行对链表的操作，将该链表的节点值往前移一位，覆盖掉删除的那个节点的值就可以了。

本人代码：



## [21. 合并两个有序链表](https://leetcode-cn.com/problems/merge-two-sorted-lists/)

### 1 原题目

将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

**示例：**

输入：1->2->4, 1->3->4

输出：1->1->2->3->4->4

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

通过比较两个链表的节点值的大小来决定谁先加入到新链表中，加入到新链表的那个链表指针要后移，继续比较，以此类推。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* mergeTwoLists(struct ListNode\* l1, struct ListNode\* l2){

struct ListNode \*ans=(struct ListNode \*)malloc(sizeof(struct ListNode));//返回去的链表的首个节点

struct ListNode \*obj=ans;//用obj来进行一系列操作

if(l1==NULL||l2==NULL){//其中一个是空链表时

if(l1==NULL&&l2==NULL){

ans=NULL;

return ans;

}

if(l1==NULL){

ans=l2;

return ans;

}

if(l2==NULL){

ans=l1;

return ans;

}

}

if(l1->val>=l2->val){//比较两个链表的首元素

obj->val=l2->val;

l2=l2->next;

}

else{

obj->val=l1->val;

l1=l1->next;

}

while(l1!=NULL&&l2!=NULL){//循环比较，直到一个为空

if(l1->val>=l2->val){

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=l2->val;

obj->next=temp;

obj=obj->next;

l2=l2->next;

}

else{

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=l1->val;

obj->next=temp;

obj=obj->next;

l1=l1->next;

}

}

if(l1==NULL&&l2==NULL)

obj->next=NULL;

if(l1==NULL)//如果while循环结束只有一个链表为空

obj->next=l2;//另一个链表剩余元素直接加到返回的链表里

if(l2==NULL)

obj->next=l1;

return ans;

}

## [83. 删除排序链表中的重复元素](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list/)

### 1 原题目

给定一个排序链表，删除所有重复的元素，使得每个元素只出现一次。

**示例 1:**

输入: 1->1->2

输出: 1->2

**示例 2:**

输入: 1->1->2->3->3

输出: 1->2->3

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

用两个指针，一个指当前节点，另一个指当前节点的下一个节点，将二者的节点值进行比较，相等的话，就将指针的指向跳过当前节点的下一个节点，指向当前节点下一节点的下一节点。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* deleteDuplicates(struct ListNode\* head){

if(head==NULL){//当所传链表为空时

struct ListNode \*l1=NULL;

return l1;

}

struct ListNode \*l1=head;//l1作为返回的链表

struct ListNode \*l2=head->next;

struct ListNode \*l3=head;

while(l3!=NULL&&l2!=NULL){//循环条件

if(l3->val==l2->val){//如果值相等

l3->next=l2->next;//指针越过此项指向下一项

l2=l2->next;

}

else{

l3=l3->next;

l2=l2->next;

}

}

return l1;

}

## [82. 删除排序链表中的重复元素 II](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii/)

### 1 原题目

给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 没有重复出现 的数字。

**示例 1:**

**输入: 1->2->3->3->4->4->5**

**输出: 1->2->5**

**示例 2:**

**输入: 1->1->1->2->3**

**输出: 2->3**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

将链表的节点值进行比较，如果相同记录下相同元素的前一个位置和相同元素结束的后一个位置，将指针跳过这一段区间就可以了。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* deleteDuplicates(struct ListNode\* head){

struct ListNode\* ans=head;//ans是最后

if(ans==NULL||ans->next==NULL)//当链表只有一个节点和无节点时

return ans;

int temp;

while(ans->next!=NULL&&ans->val==ans->next->val){//此while循环的作用是将链表开端的重复元素删掉

temp=ans->val;

while(ans!=NULL&&ans->val==temp)

ans=ans->next;

if(ans==NULL||ans->next==NULL)

return ans;

}

struct ListNode\* l1=(struct ListNode\* )malloc(sizeof(struct ListNode));//头节点，指向ans

l1->next=ans;

struct ListNode\* l2=ans;

struct ListNode\* l3=ans->next;

while(l3!=NULL){

if(l2->val==l3->val){//如果相等

while(l3!=NULL&&l2->val==l3->val){//此while循环结束后l2为找到重复出现的数字最后出现的位置

l2=l2->next;

l3=l3->next;

}

if(l3!=NULL){//重复元素最后出现的位置不在链表尾部时

l2=l2->next;

l3=l3->next;

l1->next=l2;

}

else//如果重复元素最后出现的位置在链表的尾部时

l1->next=NULL;

}

else{//不相等时，链表节点下移

l1=l1->next;

l2=l2->next;

l3=l3->next;

}

}

return ans;

}

## [203. 移除链表元素](https://leetcode-cn.com/problems/remove-linked-list-elements/)

### 1 原题目

删除链表中等于给定值 ***val***的所有节点。

**示例:**

**输入:** 1->2->6->3->4->5->6, ***val*** = 6

**输出:** 1->2->3->4->5

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

就是将节点值与所给值进行比较，相等的话就通过指针指向的变化来达到移除节点的目的。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* removeElements(struct ListNode\* head, int val){

struct ListNode \*l1=head;//返回的链表的首个节点

if(head==NULL)//当链表为空时

return l1;

if(head->next==NULL){//当链表只有一个节点时

if(head->val==val){

l1=NULL;

return l1;

}

else

return l1;

}

while(head!=NULL&&head->val==val){//从首节点开始的一段都等于所给值时

l1=head->next;

head=head->next;

if(head==NULL){//链表所有的节点值都等于所给值

l1=NULL;

return l1;

}

}

struct ListNode \*l2=l1->next;

struct ListNode \*l3=l1;

while(l2!=NULL&&l3!=NULL){//循环的条件

if(l2->val==val){

l3->next=l2->next;

l2=l3->next;

}

else{

l2=l2->next;

l3=l3->next;

}

}

return l1;

}

## [61. 旋转链表](https://leetcode-cn.com/problems/rotate-list/)

### 1 原题目

给定一个链表，旋转链表，将链表每个节点向右移动 k 个位置，其中 k 是非负数。

**示例 1:**

**输入: 1->2->3->4->5->NULL, k = 2**

**输出: 4->5->1->2->3->NULL**

**解释:**

**向右旋转 1 步: 5->1->2->3->4->NULL**

**向右旋转 2 步: 4->5->1->2->3->NULL**

**示例 2:**

**输入: 0->1->2->NULL, k = 4**

**输出: 2->0->1->NULL**

**解释:**

**向右旋转 1 步: 2->0->1->NULL**

**向右旋转 2 步: 1->2->0->NULL**

**向右旋转 3 步: 0->1->2->NULL**

**向右旋转 4 步: 2->0->1->NULL**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

将链表中的节点先存进数组里，然后进行旋转链表的操作就很容易了，我是将链表所有节点值存在一个数组中，将需要从后面移动到前面的节点值又存在另外的一个数组中

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* rotateRight(struct ListNode\* head, int k){

struct ListNode \*ans=head;//ans是返回的链表

if(head==NULL){//当所传进来的链表无节点时

ans=NULL;

return ans;

}

struct ListNode \*obj=ans;//用obj来进行操作

int arr[5000]={0},n=0,arr1[1000]={0},m=0;

while(head!=NULL){//用arr记录链表中的所有元素，

arr[n++]=head->val;//n记录链表的节点数量

head=head->next;

}

int k1=k%n,k2=k1;

while(k1>0){//arr1记录有哪些在后面的元素需要移动到前面

arr1[m++]=arr[--n];

k1--;

}

if(k2==0)//k2为0说明原链表的结构不用改变

return ans;

else

obj->val=arr1[--m];

while(m>0){//移动后面的节点到前面

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr1[--m];

obj->next=temp;

obj=obj->next;

}

while(k1<n){//前面的节点往后移k个位置

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));

temp->val=arr[k1++];

obj->next=temp;

obj=obj->next;

}

obj->next=NULL;

return ans;

}

#### 3.2 解法2（C语言已过）

先将链表头尾相连连接成一个环，然后从旧得头部开始遍历，遍历到第n - (k % n)

位置时（n为原始链表的节点数），就跳出循环，此时的这个节点就是新链表的尾部，此节点的下一个节点就为链表新的头部。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* rotateRight(struct ListNode\* head, int k){

    if(head == NULL || head -> next == NULL)//当无节点和只有一个节点时

        return head;

    struct ListNode \*h = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*new\_head = (struct ListNode \*)malloc(sizeof(struct ListNode));

    h = head;

    int n = 1;

    while(head -> next != NULL) {

        head = head -> next;

        n++;

    }

    head -> next = h;//将原始链表连接成一个环

    int m = 1;

    while(h != NULL) {

        if(m == n - (k % n))//找到新的链表尾部

            break;

        m++;

        h = h -> next;

    }

    new\_head = h -> next;//新的链表头部

    h -> next = NULL;

    return new\_head;

}

## [19. 删除链表的倒数第N个节点](https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/)

### 1 原题目

给定一个链表，删除链表的倒数第 n 个节点，并且返回链表的头结点。

**示例：**

**给定一个链表: 1->2->3->4->5, 和 n = 2.**

**当删除了倒数第二个节点后，链表变为 1->2->3->5.**

**说明：**

给定的 n 保证是有效的。

**进阶：**

你能尝试使用一趟扫描实现吗？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先第一遍遍历链表，记录出链表的总结点数，然后计算出所要删除的节点正数的位置，然后再次遍历链表，进行节点删除的操作

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* removeNthFromEnd(struct ListNode\* head, int n){

struct ListNode \*ans=head;//ans是返回的链表

struct ListNode \*p=head;

struct ListNode \*temp=(struct ListNode \*)malloc(sizeof(struct ListNode));//temp是头节点，加在head前

temp->next=head;

int num=0,i;

while(head!=NULL){

head=head->next;

num++;//记录节点的个数

}

i=num-n+1;//计算出从正数的位置

if(i==1){//当删除节点是正数第一个时

ans=ans->next;

return ans;

}

while(i>1){//找到所要删除节点的位置

p=p->next;

temp=temp->next;

i--;

}

temp->next=p->next;//删除节点

return ans;

}

## [141. 环形链表](https://leetcode-cn.com/problems/linked-list-cycle/)

### 1 原题目

给定一个链表，判断链表中是否有环。

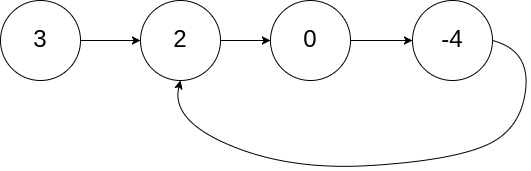
为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 pos 是 -1，则在该链表中没有环。

**示例 1：**

**输入：head = [3,2,0,-4], pos = 1**

**输出：true**

**解释：链表中有一个环，其尾部连接到第二个节点。**

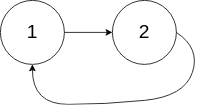


**示例 2：**

**输入：head = [1,2], pos = 0**

**输出：true**

**解释：链表中有一个环，其尾部连接到第一个节点。**



**示例 3：**

**输入：head = [1], pos = -1**

**输出：false**

**解释：链表中没有环。**



**进阶：**

你能用 O(1)（即，常量）内存解决此问题吗？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

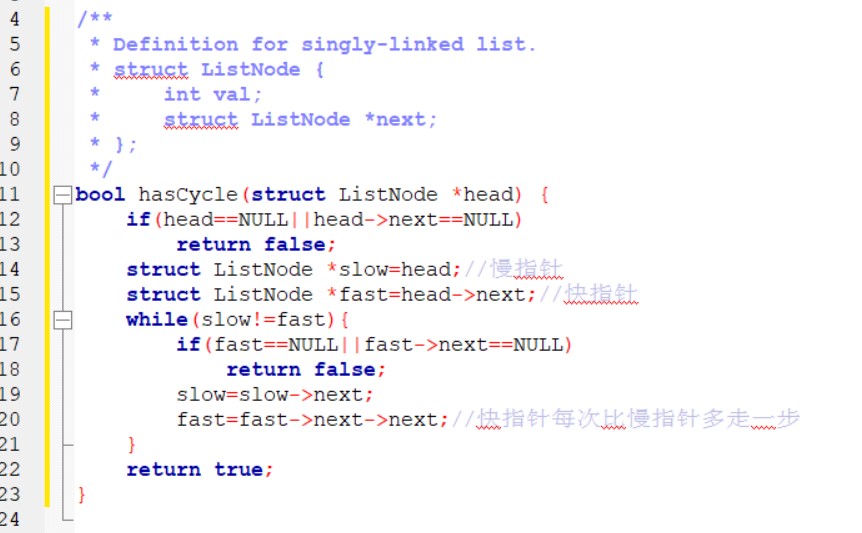
无。

### 3 解法

#### 3.1 解法1（C语言已过）

用双指针，一个快指针，一个慢指针，假如链表中没有环，那么快指针会先到达尾部，如果有环，那么快指针早晚都会和慢指针相遇，就由此来判断链表是否有环

本人代码：



## [328. 奇偶链表](https://leetcode-cn.com/problems/odd-even-linked-list/)

### 1 原题目

给定一个单链表，把所有的奇数节点和偶数节点分别排在一起。请注意，这里的奇数节点和偶数节点指的是节点编号的奇偶性，而不是节点的值的奇偶性。

请尝试使用原地算法完成。你的算法的空间复杂度应为 O(1)，时间复杂度应为 O(nodes)，nodes 为节点总数。

**示例 1:**

**输入: 1->2->3->4->5->NULL**

**输出: 1->3->5->2->4->NULL**

**示例 2:**

**输入: 2->1->3->5->6->4->7->NULL**

**输出: 2->3->6->7->1->5->4->NULL**

**说明:**

应当保持奇数节点和偶数节点的相对顺序。

链表的第一个节点视为奇数节点，第二个节点视为偶数节点，以此类推。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

用两个链表来辅助，一个存奇数位的节点值，一个存偶数位的节点值，然后将链表头尾相连就可以了

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* oddEvenList(struct ListNode\* head){

struct ListNode\* ans;//ans是返回的链表

if(head==NULL||head->next==NULL){//当传进来的链表只有一个节点或者无节点时

ans=head;

return ans;

}

struct ListNode\* odd=(struct ListNode\* )malloc(sizeof(struct ListNode));//用来构建奇数位的链表

struct ListNode\* even=(struct ListNode\* )malloc(sizeof(struct ListNode));//用来构建偶数位的链表

struct ListNode\* even1;

odd->val=head->val;//链表的第一位先放进odd来

head=head->next;

ans=odd;//将ans连接到奇数链表的头部

even->val=head->val;//链表的第二位先放进even来

even1=even;

head=head->next;

int num=3;//num记录当前进行操作的节点的位置

while(head!=NULL){//遍历链表

struct ListNode\* temp=(struct ListNode\* )malloc(sizeof(struct ListNode));

if(num%2!=0){//将所传链表的奇数位节点放进odd链表中

temp->val=head->val;

odd->next=temp;

odd=odd->next;

}

else{//将所传链表的偶数位节点放进even链表中

temp->val=head->val;

even->next=temp;

even=even->next;

}

head=head->next;

num++;

}

even->next=NULL;

odd->next=even1;//奇数链表的尾部连接到偶数链表的头部

return ans;

}

## [86. 分隔链表](https://leetcode-cn.com/problems/partition-list/)

### 1 原题目

给定一个链表和一个特定值 x，对链表进行分隔，使得所有小于 x 的节点都在大于或等于 x 的节点之前。

你应当保留两个分区中每个节点的初始相对位置。

**示例:**

**输入: head = 1->4->3->2->5->2, x = 3**

**输出: 1->2->2->4->3->5**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

用两个链表来辅助，一个存大于等于所给值的的节点值，一个存小于所给值的节点值，然后在分情况将链表连接起来就可以了

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* partition(struct ListNode\* head, int x){

struct ListNode\* ans;//ans是返回的链表

if(head==NULL||head->next==NULL){//当所传链表只有一个节点或无节点时

ans=head;

return ans;

}

struct ListNode\* min=(struct ListNode\* )malloc(sizeof(struct ListNode));//存小于所给值的节点

struct ListNode\* max=(struct ListNode\* )malloc(sizeof(struct ListNode));//存大于等于所给值的节点

struct ListNode\* max1=max;

struct ListNode\* min1=min;

int num\_min=0,num\_max=0;//记录两个链表中节点的数量

while(head!=NULL){//遍历一遍链表

struct ListNode\* temp=(struct ListNode\* )malloc(sizeof(struct ListNode));

if(head->val>=x){//存大于等于所给值的节点

if(num\_max==0)

max->val=head->val;

else{

temp->val=head->val;

max->next=temp;

max=max->next;

}

num\_max++;

}

else{//存小于所给值的节点值

if(num\_min==0)

min->val=head->val;

else{

temp->val=head->val;

min->next=temp;

min=min->next;

}

num\_min++;

}

head=head->next;

}

if(num\_min==0){//当没有小于所给值的节点时

ans=max1;

max->next=NULL;

}

if(num\_max==0){//当没有大于等于所给值的节点时

ans=min1;

min->next=NULL;

}

if(num\_max!=0&&num\_min!=0){

ans=min1;

min->next=max1;

max->next=NULL;

}

return ans;

}

## [147. 对链表进行插入排序](https://leetcode-cn.com/problems/insertion-sort-list/)

### 1 原题目

对链表进行插入排序。

从第一个元素开始，该链表可以被认为已经部分排序。

每次迭代时，从输入数据中移除一个元素,并原地将其插入到已排好序的链表中。

**插入排序算法：**

插入排序是迭代的，每次只移动一个元素，直到所有元素可以形成一个有序的输出列表。

每次迭代中，插入排序只从输入数据中移除一个待排序的元素，找到它在序列中适当的位置，并将其插入。

重复直到所有输入数据插入完为止。

**示例 1：**

**输入: 4->2->1->3**

**输出: 1->2->3->4**

**示例 2：**

**输入: -1->5->3->4->0**

**输出: -1->0->3->4->5**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

找到每次需要插入排序的那个节点，然后在对链表从头进行遍历找出插入的位置，最后进行节点插入操作。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* insertionSortList(struct ListNode\* head){

    if(head == NULL || head -> next == NULL)//当无节点或只有一个节点时

        return head;

    struct ListNode \*Thead = (struct ListNode \*)malloc(sizeof(struct ListNode));//头节点

    struct ListNode \*a = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*b = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*c = (struct ListNode \*)malloc(sizeof(struct ListNode));

    Thead -> next = head;

    a = head;

    b = Thead;

    while(a && a -> next && a -> val <= a -> next -> val)//最开始如果有一段是有升序排序的就跳过去

        a = a -> next;

    c = a -> next;

    while(c) {//c节点为当前要进行插入排序的节点

        a -> next = c -> next;//因为c节点被拿出了，所以a要与c后面的点连接上

        while(b -> next && b -> next -> val < c -> val)//找到c要插入的位置

            b = b -> next;

        c -> next = b -> next;

        b -> next = c;

        b = Thead;//b重置以便下次寻找插入位置

        while(a && a -> next && a -> val <= a -> next -> val)//a与c后面的点重接后如果有一段升序排序就跳过

            a = a -> next;

        c = a -> next;

    }

    return Thead -> next;

}

## [148. 排序链表](https://leetcode-cn.com/problems/sort-list/)

### 1 原题目

在 O(n log n) 时间复杂度和常数级空间复杂度下，对链表进行排序。。

**示例 1：**

**输入: 4->2->1->3**

**输出: 1->2->3->4**

**示例 2：**

**输入: -1->5->3->4->0**

**输出: -1->0->3->4->5**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

找到每次需要插入排序的那个节点，然后在对链表从头进行遍历找出插入的位置，最后进行节点插入操作。（时间复杂度可能不符合要求）

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* insertionSortList(struct ListNode\* head){

    if(head == NULL || head -> next == NULL)//当无节点或只有一个节点时

        return head;

    struct ListNode \*Thead = (struct ListNode \*)malloc(sizeof(struct ListNode));//头节点

    struct ListNode \*a = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*b = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*c = (struct ListNode \*)malloc(sizeof(struct ListNode));

    Thead -> next = head;

    a = head;

    b = Thead;

    while(a && a -> next && a -> val <= a -> next -> val)//最开始如果有一段是有升序排序的就跳过去

        a = a -> next;

    c = a -> next;

    while(c) {//c节点为当前要进行插入排序的节点

        a -> next = c -> next;//因为c节点被拿出了，所以a要与c后面的点连接上

        while(b -> next && b -> next -> val < c -> val)//找到c要插入的位置

            b = b -> next;

        c -> next = b -> next;

        b -> next = c;

        b = Thead;//b重置以便下次寻找插入位置

        while(a && a -> next && a -> val <= a -> next -> val)//a与c后面的点重接后如果有一段升序排序就跳过

            a = a -> next;

        c = a -> next;

    }

    return Thead -> next;

}

## [23. 合并K个排序链表](https://leetcode-cn.com/problems/merge-k-sorted-lists/)

### 1 原题目

合并 k 个排序链表，返回合并后的排序链表。请分析和描述算法的复杂度。

**示例:**

**输入:**

**[**

**1->4->5,**

**1->3->4,**

**2->6**

**]**

**输出: 1->1->2->3->4->4->5->6**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

将其他链表的节点插入到第一个链表上，遍历其余链表的每一个节点，然后在对第一个链表从头进行遍历找出插入的位置，最后进行节点插入操作。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* mergeKLists(struct ListNode\*\* lists, int listsSize){

    if(listsSize == 0)

        return 0;

    int i;

    struct ListNode \*temp = (struct ListNode \*)malloc(sizeof(struct ListNode));//每次找插入位置的节点

    struct ListNode \*next = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*head = (struct ListNode \*)malloc(sizeof(struct ListNode));//头节点

    head -> next = lists[0];

    temp = head;

    for(i = 1; i < listsSize; i++) {//遍历其余链表

        while(lists[i]) {//遍历当前链表的每一个节点

        while(temp -> next && temp -> next -> val <= lists[i] -> val)//找到插入的位置

            temp = temp -> next;

        next = lists[i] -> next;//记录下此节点的下一个节点

        lists[i] -> next = temp -> next;

        temp -> next = lists[i];

        lists[i] = next;//将此节点的下一个位置赋给它

        temp = head;//重置temp以便下次继续寻找插入位置

        }

    }

    return head -> next;

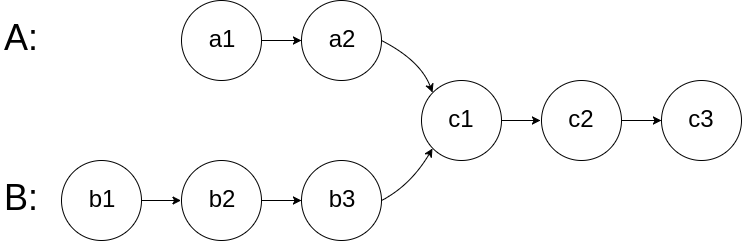
}

## [160. 相交链表](https://leetcode-cn.com/problems/intersection-of-two-linked-lists/)

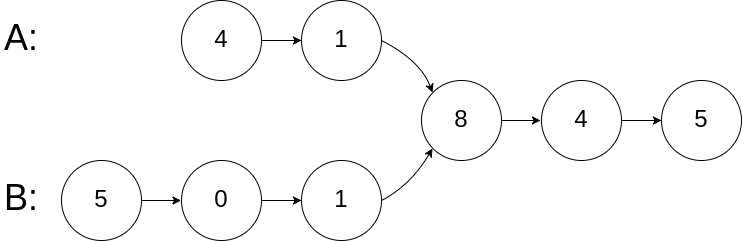
### 1 原题目

编写一个程序，找到两个单链表相交的起始节点。

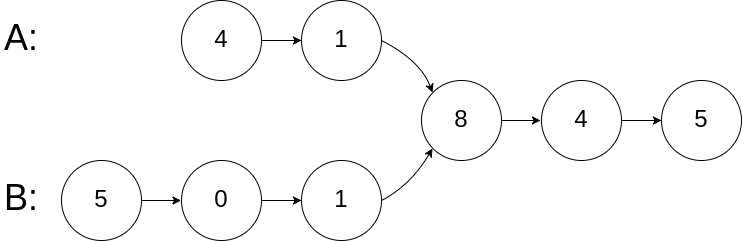
如下面的两个链表：



在节点 c1 开始相交。



**示例 1：**

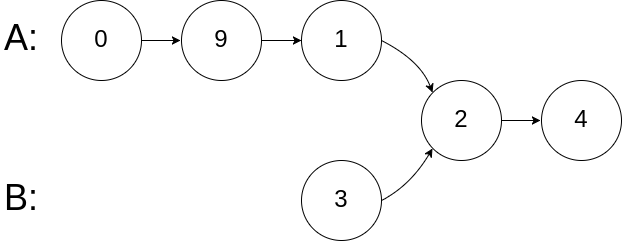


**输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3**

**输出：Reference of the node with value = 8**

**输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。**

**示例 2：**

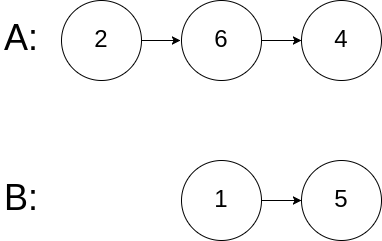


**输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1**

**输出：Reference of the node with value = 2**

**输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。**

**示例 3：**



**输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2**

**输出：null**

**输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。**

**解释：这两个链表不相交，因此返回 null。**

**注意：**

如果两个链表没有交点，返回 null.

在返回结果后，两个链表仍须保持原有的结构。

可假定整个链表结构中没有循环。

程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

两个链表都先走自己的部分，如果自己的走完了，那么就去走对方的路，这样二者所走的路程就一样了，所以要么在交点处相遇，要么就是无交点（建议大家自己画图走一下）

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode \*getIntersectionNode(struct ListNode \*headA, struct ListNode \*headB) {

    if(headA == NULL || headB == NULL)

        return NULL;

    struct ListNode \*PA = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*PB = (struct ListNode \*)malloc(sizeof(struct ListNode));

    PA = headA;

    PB = headB;

    while(PA != PB) {

        PA = (PA == NULL) ? headB : PA -> next;//先走自己的路再走另一个链表的路

        PB = (PB == NULL) ? headA : PB -> next;//这样二者走过的长度是一样的，如果有交点则一定会在交点处相遇

    }

    return PA;

}

## [234. 回文链表](https://leetcode-cn.com/problems/palindrome-linked-list/)

### 1 原题目

请判断一个链表是否为回文链表。

**示例 1:**

**输入: 1->2**

**输出: false**

**示例 2:**

**输入: 1->2->2->1**

**输出: true**

**进阶：**

你能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题？

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

我的做法是先将原链表的原始的样子存在一个链表中，在建立一个链表将原链表翻转过来，然后遍历两个链表，判断每一个对应的值是否相等，全都相等就是回文链表，出现不相等的就说明不是回文链表

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

bool isPalindrome(struct ListNode\* head){

if(head==NULL||head->next==NULL)//链表只有一个节点和无节点时

return true;

if(head->next->next==NULL){//链表有两个节点时

if(head->val==head->next->val)

return true;

else

return false;

}

struct ListNode\* l1=head;

struct ListNode\* l2=head->next;

struct ListNode\* l3=head->next->next;

struct ListNode\* p1=(struct ListNode\*)malloc(sizeof(struct ListNode));//p1存链表的正常顺序

struct ListNode\* p2=p1;

p1->val=head->val;

head=head->next;

while(head!=NULL){//链表正常顺序存进p1

struct ListNode\* temp=(struct ListNode\*)malloc(sizeof(struct ListNode));

temp->val=head->val;

p1->next=temp;

p1=p1->next;

head=head->next;

}

p1->next=NULL;

l1->next=NULL;

while(l3!=NULL){//链表翻转

l2->next=l1;

l1=l2;

l2=l3;

l3=l3->next;

}

l2->next=l1;//链表翻转后的首个节点是l2

while(p2!=NULL){//两个链表进行比较

if(p2->val!=l2->val)

return false;

p2=p2->next;

l2=l2->next;

}

return true;

}

## [24. 两两交换链表中的节点](https://leetcode-cn.com/problems/swap-nodes-in-pairs/)

### 1 原题目

给定一个链表，两两交换其中相邻的节点，并返回交换后的链表。

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

**示例:**

**给定 1->2->3->4, 你应该返回 2->1->4->3.**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

直接进行指针指向的改变，大家看不懂的话可以自己在纸上画出一个链表，走一遍代码中while循环中的内容。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* swapPairs(struct ListNode\* head){

    struct ListNode \*Thead = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*a = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*b = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*c = (struct ListNode \*)malloc(sizeof(struct ListNode));

    Thead -> next = head;

    c = Thead;

    while(c -> next && c -> next -> next) {

        a = c -> next;

        b = c -> next -> next;

        c -> next = b;

        a -> next = b -> next;

        b -> next = a;

        c = c -> next -> next;

    }

    return Thead -> next;

}

## [25. K 个一组翻转链表](https://leetcode-cn.com/problems/reverse-nodes-in-k-group/)

### 1 原题目

给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。

k 是一个正整数，它的值小于或等于链表的长度。

如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

**示例 :**

**给定这个链表：1->2->3->4->5**

**当 k = 2 时，应当返回: 2->1->4->3->5**

**当 k = 3 时，应当返回: 3->2->1->4->5**

**说明 :**

你的算法只能使用常数的额外空间。

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（java已过）

将链表反转还算比较简单（改变链表指针的指向就可以），难就难在如何反转后连接起来。在这里我们可以利用递归的思想来进行链表反转后的连接。（具体请看代码和注释）

此题比较好的讲解：<https://mp.weixin.qq.com/s?__biz=MzAxODQxMDM0Mw==&mid=2247484597&idx=1&sn=c603f1752e33cb2701e371d84254aee2&chksm=9bd7fabdaca073abd512d8fff18016c9092ede45fed65c307852c65a2026d8568ee294563c78&scene=0&xtrack=1&key=2bdaca4c91627f7e2e7b574a2d20eebd54cc20aff5e23f5b589dcfb9f6c2db18da7f1b8ad673c73cf1cc005ed6ed4ba29e22b5b97bdf86f1d808333e5827023d1c7bda3ef3ba127273e3657538d8fa4d&ascene=1&uin=MTA1MzA1MTU3NA%3D%3D&devicetype=Windows+10&version=62060841&lang=zh_CN&pass_ticket=8jJWl14qBiRZl3eOMLjtsbH5mGwWdgBWNSFmwjTDu77Fs%2Fwb3kBfdqrCoCpys51O>

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public ListNode reverse(ListNode a,ListNode b) {

        ListNode prev = null,curr = a,next = a;

        while (curr != b) {//反转链表

            next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

        }

        return prev;//返回反转后的头结点

    }

    public ListNode reverseKGroup(ListNode head, int k) {

        if(head == null)

            return head;

        ListNode a,b;

        a = head;

        b = head;

        for(int i = 0; i < k; i++) {

            if(b == null)//当不满足k个时，不需要反转

                return head;

            b = b.next;

        }

        ListNode newHead = reverse(a,b);//先反转前k个

        a.next = reverseKGroup(b,k);//然后通过递归继续反转后面的链表，并将链表连接起来

        return newHead;

    }

}

## [2. 两数相加](https://leetcode-cn.com/problems/add-two-numbers/)

### 1 原题目

给出两个 非空 的链表用来表示两个非负的整数。其中，它们各自的位数是按照 逆序 的方式存储的，并且它们的每个节点只能存储 一位 数字。

如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

您可以假设除了数字 0 之外，这两个数都不会以 0 开头。

**示例：**

**输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)**

**输出：7 -> 0 -> 8**

**原因：342 + 465 = 807**

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

开辟三个数组，然后用两个数组分别存储两个链表的节点值，剩下的一个数组存储两个数加起来后的结果，相加的运算就直接从两个数组的头开始相加直到长的那个数组加完，需要判断一下最后加完时是否整体的长度比最长的数字进了一位。最后通过这个存储结果的数组中的值构建出所要返回的链表。

本人代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* addTwoNumbers(struct ListNode\* l1, struct ListNode\* l2){

int arr1[10000]={0},arr2[10000]={0},res[10000]={0},a1=0,a2=0;//res数组是记录相加的结果

while(l1!=NULL){//arr1数组存储l1链表的节点值

arr1[a1++]=l1->val;

l1=l1->next;

}

while(l2!=NULL){//arr2数组存储l2链表的节点值

arr2[a2++]=l2->val;

l2=l2->next;

}

int carry=0,i,len;//carry记录进位的数

len=a1>a2?a1:a2;//len记录长的数字的位数

for(i=0;i<len+1;i++){//小于len+1是为了防止两个数相加后总的整个长度比长的数字的位数加1位

if(arr1[i]+arr2[i]+carry>=10){//判断是否有进位

res[i]=(arr1[i]+arr2[i]+carry)%10;//记录当前位的数字

carry=(arr1[i]+arr2[i]+carry)/10;//记录进位数

}

else{

res[i]=arr1[i]+arr2[i]+carry;

carry=0;//无进位数时，carry要变为0，等待重新记录下次进位值

}

}

if(res[len]!=0)//判断是否总长度加1

len=len+1;

struct ListNode \*head=(struct ListNode \*)malloc(sizeof(struct ListNode));

struct ListNode \*p=head;

head->val=res[0];

for(i=1;i<len;i++){//构建新的返回链表

struct ListNode \*q=(struct ListNode \*)malloc(sizeof(struct ListNode));

q->val=res[i];

p->next=q;

p=p->next;

}

p->next=NULL;

return head;

}

#### 3.2 解法2（C语言已过）

因为题目里所给的链表的两个数和让我们返回结果的链表都是数字倒过来存储的，所以我们直接加法运算，利用链表节点直接记算进位和计算当前位的值来计算出最后的结果，具体请看代码和注释。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     struct ListNode \*next;

 \* };

 \*/

struct ListNode\* addTwoNumbers(struct ListNode\* l1, struct ListNode\* l2){

    struct ListNode \*curr = (struct ListNode \*)malloc(sizeof(struct ListNode));

    struct ListNode \*res = (struct ListNode \*)malloc(sizeof(struct ListNode));//返回结果链表的头指针

    res = curr;

    int carry = 0;//记录进位

    while(l1 != NULL || l2 != NULL) {//循环条件

        int x = 0, y = 0;

        if(l1 != NULL) {

            x = l1 -> val;

            l1 = l1 -> next;

        }

        if(l2 != NULL){

            y = l2 -> val;

            l2 = l2 -> next;

        }

        curr -> val = (x + y + carry) % 10;//存储当前节点的值

        carry = (x + y + carry) / 10;//计算进位

        if(l1 == NULL && l2 == NULL)//两个链表都遍历完就跳出循环

            break;

        struct ListNode \*temp = (struct ListNode \*)malloc(sizeof(struct ListNode));

        curr -> next = temp;

        curr = curr -> next;

    }

    if(carry != 0) {//如果还有进位

        struct ListNode \*temp = (struct ListNode \*)malloc(sizeof(struct ListNode));

        temp -> val = carry;

        curr -> next = temp;

        curr = curr -> next;

    }

    curr -> next = NULL;

    return res;

}

## [232. 用栈实现队列](https://leetcode-cn.com/problems/implement-queue-using-stacks/)

### 1 原题目

使用栈实现队列的下列操作：

push(x) -- 将一个元素放入队列的尾部。

pop() -- 从队列首部移除元素。

peek() -- 返回队列首部的元素。

empty() -- 返回队列是否为空。

**示例:**

MyQueue queue = new MyQueue();

queue.push(1);

queue.push(2);

queue.peek(); // 返回 1

queue.pop(); // 返回 1

queue.empty(); // 返回 false

**说明:**

你只能使用标准的栈操作 -- 也就是只有 push to top, peek/pop from top, size, 和 is empty 操作是合法的。

你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。

假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/implement-queue-using-stacks>

### 2 基础知识

#### 2.1数据结构

栈（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

（本代码不知道是否符合题目中的只能使用标准的栈操作这一项）

本人代码：

typedef struct {

int base;//栈底指针

int top;//栈顶指针

int arr[1000];

} MyQueue;

/\*\* Initialize your data structure here. \*/

MyQueue\* myQueueCreate() {

MyQueue\* obj=(MyQueue \*)malloc(sizeof(MyQueue));

obj->base=0;

obj->top=0;

memset(obj->arr,0,sizeof(obj->arr));//数组初始化

return obj;

}

/\*\* Push element x to the back of queue. \*/

void myQueuePush(MyQueue\* obj, int x) {

obj->arr[obj->top]=x;

obj->top++;

}

/\*\* Removes the element from in front of queue and returns that element. \*/

int myQueuePop(MyQueue\* obj) {

int res=obj->arr[obj->base];

obj->base++;

return res;

}

/\*\* Get the front element. \*/

int myQueuePeek(MyQueue\* obj) {

return obj->arr[obj->base];

}

/\*\* Returns whether the queue is empty. \*/

bool myQueueEmpty(MyQueue\* obj) {

if(obj->base==obj->top)

return true;

return false;

}

void myQueueFree(MyQueue\* obj) {

free(obj);

}

/\*\*

\* Your MyQueue struct will be instantiated and called as such:

\* MyQueue\* obj = myQueueCreate();

\* myQueuePush(obj, x);

\* int param\_2 = myQueuePop(obj);

\* int param\_3 = myQueuePeek(obj);

\* bool param\_4 = myQueueEmpty(obj);

\* myQueueFree(obj);

\*/

## [剑指 Offer 09. 用两个栈实现队列](https://leetcode-cn.com/problems/yong-liang-ge-zhan-shi-xian-dui-lie-lcof/)

### 1 原题目

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1 )

**示例 1：**

输入：

["CQueue","appendTail","deleteHead","deleteHead"]

[[],[3],[],[]]

输出：[null,null,3,-1]

**示例 2：**

输入：

["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]

[[],[],[5],[2],[],[]]

输出：[null,-1,null,null,5,2]

**提示：**

1 <= values <= 10000

最多会对 appendTail、deleteHead 进行 10000 次调用

### 2 基础知识

栈（解法一用到）

队列（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

插入元素：

sa 直接插入元素

删除元素：

如果 ta 为空，则将 sa 里的所有元素弹出插入到ta里

如果 ta 仍为空，则返回 -1，否则从 ta 弹出一个元素并返回

本人代码：

|  |
| --- |
| class CQueue {  public:      stack<int> sa;//插入栈      stack<int> ta;//删除栈      CQueue() {      }        void appendTail(int value) {          sa.push(value);//元素直接插入到插入栈      }        int deleteHead() {          if(ta.empty()) {//删除栈为空              while(!sa.empty()) {//将插入栈元素存进删除栈                  int temp = sa.top();                  sa.pop();                  ta.push(temp);              }          }          if(ta.empty())              return -1;          int res = ta.top();          ta.pop();          return res;      }  };  /\*\*   \* Your CQueue object will be instantiated and called as such:   \* CQueue\* obj = new CQueue();   \* obj->appendTail(value);   \* int param\_2 = obj->deleteHead();   \*/ |

## [225. 用队列实现栈](https://leetcode-cn.com/problems/implement-stack-using-queues/)

### 1 原题目

使用队列实现栈的下列操作：

push(x) -- 元素 x 入栈

pop() -- 移除栈顶元素

top() -- 获取栈顶元素

empty() -- 返回栈是否为空

**注意:**

你只能使用队列的基本操作-- 也就是 push to back, peek/pop from front, size, 和 is empty 这些操作是合法的。

你所使用的语言也许不支持队列。 你可以使用 list 或者 deque（双端队列）来模拟一个队列 , 只要是标准的队列操作即可。

你可以假设所有操作都是有效的（例如, 对一个空的栈不会调用 pop 或者 top 操作）。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/implement-stack-using-queues>

### 2 基础知识

#### 2.1数据结构

队列（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

队列的性质是先进先出，栈的性质是先进后出。我们在利用队列实现栈时，可以在压入元素的时候进行如下的操作进可以实现利用队列实现栈：先将当前元素压进队列中，然后在将队列中前n – 1个元素重新加入队列中（n为当前队列中元素的个数），这样就实现了利用队列实现先进后出的效果。

本人代码：

class MyStack {

    queue<int>q\_s;//队列

public:

    /\*\* Initialize your data structure here. \*/

    MyStack() {

    }

    /\*\* Push element x onto stack. \*/

    void push(int x) {

        q\_s.push(x);//将元素存进队列中

        int i;

        for(i = 1; i <= q\_s.size() - 1; i++) {//将前q\_s.size() - 1个元素重新存进队列中

            q\_s.push(q\_s.front());

            q\_s.pop();

        }

    }

    /\*\* Removes the element on top of the stack and returns that element. \*/

    int pop() {

        int res = q\_s.front();

        q\_s.pop();

        return res;

    }

    /\*\* Get the top element. \*/

    int top() {

        return q\_s.front();

    }

    /\*\* Returns whether the stack is empty. \*/

    bool empty() {

        return q\_s.empty();

    }

};

/\*\*

 \* Your MyStack object will be instantiated and called as such:

 \* MyStack\* obj = new MyStack();

 \* obj->push(x);

 \* int param\_2 = obj->pop();

 \* int param\_3 = obj->top();

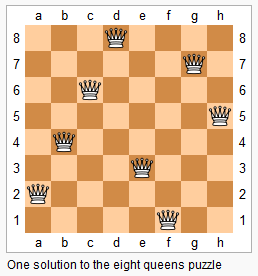
 \* bool param\_4 = obj->empty();

 \*/

## [51. N皇后](https://leetcode-cn.com/problems/n-queens/)

### 1 原题目

n 皇后问题研究的是如何将 n 个皇后放置在 n×n 的棋盘上，并且使皇后彼此之间不能相互攻击。



上图为 8 皇后问题的一种解法。

给定一个整数 n，返回所有不同的 n 皇后问题的解决方案。

每一种解法包含一个明确的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.' 分别代表了皇后和空位。

**示例:**

输入: 4

输出: [

[".Q..", // 解法 1

"...Q",

"Q...",

"..Q."],

["..Q.", // 解法 2

"Q...",

"...Q",

".Q.."]

]

解释: 4 皇后问题存在两个不同的解法。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/n-queens>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

运用回溯算法做，将三维数组全部初始化成英文句号，然后如果找到一种正确的放法就将这种放法的坐标改成字母Q。

N皇后问题参考链接：<https://www.cnblogs.com/dongsheng/archive/2012/08/19/2646641.html>

本人代码：

/\*\*

 \* Return an array of arrays of size \*returnSize.

 \* The sizes of the arrays are returned as \*returnColumnSizes array.

 \* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

 \*/

//map记录皇后放置的位置

int num,map[1000];

void dfs(int k,int n,char \*\*\*res){

    if(k == n+1){//当k等于n+1时就说明找到了一种摆放方式

        int x,y = 0;

        for(x = 1; x < k; x++)

            res[num][y++][map[x]-1] = 'Q';

        num++;//num是记录n皇后问题的摆放方式的数量

        return;//回退

    }

    int i,j,flag;

    for(i = 1; i <= n; i++){

        map[k] = i;//尝试当前皇后所放置的位置

        flag = 1;

        for(j = 1; j < k; j++){//当前皇后的位置与之前所有皇后的位置进行判断

            if(map[j] == i || i - k == map[j] - j || i + k == map[j] + j){//满足任意一个就说明皇后摆放的位置不正确

                flag = 0;

                break;

            }

        }

        if(flag == 1)//皇后位置正确

            dfs(k+1,n,res);//递归调用判断下一个皇后的位置

    }

}

char \*\*\* solveNQueens(int n, int\* returnSize, int\*\* returnColumnSizes) {

    memset(map,0,sizeof(map));//初始化

    num = 0;//记录有多少种解法

    int i,j,k;

    char \*\*\*res;

    res = (char \*\*\*)malloc(sizeof(char \*\*)\*1000);//开辟三维数组

    for(i = 0;i < 1000;i++) {

        res[i] = (char \*\*)malloc(sizeof(char\*) \* (n+1));

        for(j = 0;j < n;j++) {

            res[i][j] = (char \*)malloc(sizeof(char) \* (n+1));

            for(k = 0;k < n;k++){

                res[i][j][k] = '.';//数组中都初始化为'.'

                res[i][j][k+1] = '\0';//因为是字符串，所以最后要加\0

            }

        }

    }

    dfs(1,n,res);

    \*returnSize = num;//解法的种数

    \*returnColumnSizes = (int \*)malloc(sizeof(int) \* num);

    for(i=0;i<num;i++)

        (\*returnColumnSizes)[i] = n;//记录子二维数组的行数

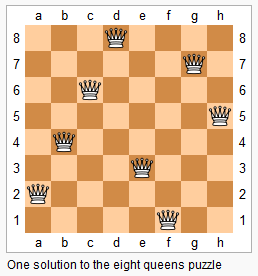
    return res;

}

## [52. N皇后 II](https://leetcode-cn.com/problems/n-queens-ii/)

### 1 原题目

n 皇后问题研究的是如何将 n 个皇后放置在 n×n 的棋盘上，并且使皇后彼此之间不能相互攻击。



上图为 8 皇后问题的一种解法。

给定一个整数 n，返回 n 皇后不同的解决方案的数量。

**示例:**

输入: 4

输出: 2

解释: 4皇后问题存在如下两个不同的解法。

[

 [".Q..",  // 解法 1

  "...Q",

  "Q...",

  "..Q."],

 ["..Q.",  // 解法 2

  "Q...",

  "...Q",

  ".Q.."]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/n-queens-ii>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法。（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

运用回溯算法做，关于回溯算法本人也是有点迷糊，没法讲解。

N皇后问题参考链接：<https://www.cnblogs.com/dongsheng/archive/2012/08/19/2646641.html>

本人代码：

static int num,map[10000];//全局变量

void dfs(int k,int n){

if(k==n+1){//当k等于n+1时就说明找到了一种摆放方式

num++;//num是记录n皇后问题的摆放方式的数量

return;//回退

}

int i,j,flag;

for(i=1;i<=n;i++){

map[k]=i;//尝试当前皇后所放置的位置

flag=1;

for(j=1;j<k;j++){//当前皇后的位置与之前所有皇后的位置进行判断

if(map[j]==i||i-k==map[j]-j||i+k==map[j]+j){//满足任意一个就说明皇后摆放的位置不正确

flag=0;

break;

}

}

if(flag==1)//皇后位置正确

dfs(k+1,n);//递归调用判断下一个皇后的位置

}

}

int totalNQueens(int n){

num=0;//初始化

memset(map,0,sizeof(map));//初始化

dfs(1,n);

return num;

}

## [22. 括号生成](https://leetcode-cn.com/problems/generate-parentheses/)

### 1 原题目

给出 n 代表生成括号的对数，请你写出一个函数，使其能够生成所有可能的并且有效的括号组合。

例如，给出 n = 3，生成结果为：

[

"((()))",

"(()())",

"(())()",

"()(())",

"()()()"

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/generate-parentheses>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

利用回溯的思想，进行括号的生成，具体请看代码的注释。

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

本人代码：

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int col,row;

//left是左括号用了几个，right是右括号用了几个

void gen(int left,int right,int n,char \*\*str) {

if(left==n&&right==n) {//左右括号都用完时

int i;

str[row][col]='\0';//字符串结束标志

row++;//继续在下一行存下一种情况

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

str[row][i]=str[row-1][i];//将这一种情况存到下一行中，因为递归回退算下一种情况是在上一种的基础上进行的

return ;

}

if(left<n) {//左括号没用完时

str[row][col++]='(';//左括号存入

gen(left+1,right,n,str);//左括号加1，递归

col=col-1;//当递归退出来时，进入下一次尝试，列数减1

}

if(left>right&&right<n) {//右括号没用完并且左括号个数大于

str[row][col++]=')';//右括号存入

gen(left,right+1,n,str);//右括号加1，递归

col=col-1;//当递归退出来时，进入下一次尝试，列数减1

}

}

char \*\* generateParenthesis(int n, int\* returnSize){

int i;

char \*\*str =(char \*\*)malloc(sizeof(char\*)\*5000);//开辟一个二维数组，有5000行

for(i=0; i<5000; i++)

str[i]=(char \*)malloc(sizeof(char)\*5000);//每一行5000列

col=0;//记录列

row=0;//记录行

gen(0,0,n,str);

\*returnSize=row;//返回行数

return str;

}

## [37. 解数独](https://leetcode-cn.com/problems/sudoku-solver/)

### 1 原题目

编写一个程序，通过已填充的空格来解决数独问题。

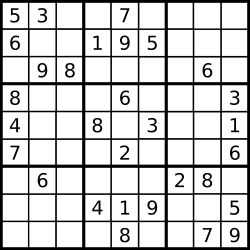
一个数独的解法需遵循如下规则：

数字 1-9 在每一行只能出现一次。

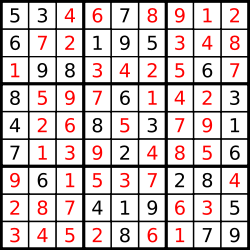
数字 1-9 在每一列只能出现一次。

数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

空白格用 '.' 表示。



一个数独。



答案被标成红色。

**Note:**

给定的数独序列只包含数字 1-9 和字符 '.' 。

你可以假设给定的数独只有唯一解。

给定数独永远是 9x9 形式的。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/sudoku-solver>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

利用回溯的思想，对数独每一个空白进行填入数字的尝试，具体请看代码的注释。

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

本人代码：

bool isok(char a,int row,int col,char \*\*str) {

int i,j;

for(j=0;j<9;j++)//判断当前行是否有重复元素

if(str[row][j]==a)

return false;

for(i=0;i<9;i++)//判断当前列是否有重复元素

if(str[i][col]==a)

return false;

if(row>=0&&row<=2) {

if(col>=0&&col<=2) {//判断第一个格子是否有重复元素

for(i=0;i<=2;i++)

for(j=0;j<=2;j++)

if(str[i][j]==a)

return false;

}

if(col>=3&&col<=5) {//判断第二个格子是否有重复元素

for(i=0;i<=2;i++)

for(j=3;j<=5;j++)

if(str[i][j]==a)

return false;

}

if(col>=6&&col<=8) {//判断第三个格子是否有重复元素

for(i=0;i<=2;i++)

for(j=6;j<=8;j++)

if(str[i][j]==a)

return false;

}

}

if(row>=3&&row<=5) {

if(col>=0&&col<=2) {//判断第四个格子是否有重复元素

for(i=3;i<=5;i++)

for(j=0;j<=2;j++)

if(str[i][j]==a)

return false;

}

if(col>=3&&col<=5) {//判断第五个格子是否有重复元素

for(i=3;i<=5;i++)

for(j=3;j<=5;j++)

if(str[i][j]==a)

return false;

}

if(col>=6&&col<=8) {//判断第六个格子是否有重复元素

for(i=3;i<=5;i++)

for(j=6;j<=8;j++)

if(str[i][j]==a)

return false;

}

}

if(row>=6&&row<=8) {

if(col>=0&&col<=2) {//判断第七个格子是否有重复元素

for(i=6;i<=8;i++)

for(j=0;j<=2;j++)

if(str[i][j]==a)

return false;

}

if(col>=3&&col<=5) {//判断第八个格子是否有重复元素

for(i=6;i<=8;i++)

for(j=3;j<=5;j++)

if(str[i][j]==a)

return false;

}

if(col>=6&&col<=8) {//判断第九个格子是否有重复元素

for(i=6;i<=8;i++)

for(j=6;j<=8;j++)

if(str[i][j]==a)

return false;

}

}

return true;

}

bool solve(char \*\*board) {

int i,j;

char k;

for(i=0;i<9;i++)

for(j=0;j<9;j++) {

if(board[i][j]=='.') {//找到数独中那个没数字的点

for(k='1';k<='9';k++) {//每个数字进行尝试

if(isok(k,i,j,board)) {

board[i][j]=k;//尝试成功就放入进去

if(solve(board))//下一次的递归调用

return true;

else//如果不行的话

board[i][j]='.';//要将它还原成'.'，因为之前填入了数字

}

}

return false;

}

}

return true;

}

void solveSudoku(char\*\* board, int boardSize, int\* boardColSize){

solve(board);

}

## [79. 单词搜索](https://leetcode-cn.com/problems/word-search/)

### 1 原题目

给定一个二维网格和一个单词，找出该单词是否存在于网格中。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

**示例:**

board =

[

['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']

]

给定 word = "ABCCED", 返回 true.

给定 word = "SEE", 返回 true.

给定 word = "ABCB", 返回 false.

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/word-search

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

首先先遍历一遍二维数组，如果二维数组的元素等于所给单词的首字母，那么就开始进行单词的查找，其中单词的查找要用到回溯算法，具体大家看代码以及注释吧。

本人代码：

//i和j是二维数组的行和列的坐标，k是记录应该找单词的哪个字母，len是单词的长度，row和col记录的是二维数组的总行和列数

bool dfs(char \*\*board,char \*word,int i,int j,int k,int len,int row,int col) {

int next[4][2]={0,1,1,0,0,-1,-1,0};//往右，下，左，上移动一个位置的坐标

int tx,ty,z;

if(k==len)//当找到所给单词时

return true;

for(z=0;z<4;z++) {//进行单词的搜索，右下左上的顺序寻找

tx=i+next[z][0];

ty=j+next[z][1];

if(tx<0||ty<0||tx>row-1||ty>col-1)//超出二维数组界限时

continue;

if(board[tx][ty]!='\*'&&board[tx][ty]==word[k]) {//当二维数组此位置没被标记并且等于单词的下一个字母时

board[tx][ty]='\*';//将二维数组此位置进行标记

if(dfs(board,word,tx,ty,k+1,len,row,col))//继续进行下一次的深度优先搜索

return true;

board[tx][ty]=word[k];//搜索失败时，还原此位置原来的字母

}

}

return false;

}

bool exist(char\*\* board, int boardSize, int\* boardColSize, char \* word){

if(strlen(word)==0)//当所给单词长度为0

return true;

int i,j;

for(i=0;i<boardSize;i++) {//遍历二维数组

for(j=0;j<\* boardColSize;j++) {

if(board[i][j]==word[0]) {//二维数组元素与单词首字母相同

board[i][j]='\*';//做标记，代表二维数组中的此字母已被用

if(dfs(board,word,i,j,1,strlen(word),boardSize,\* boardColSize))//深度优先搜索

return true;

board[i][j]=word[0];//当发现此字母开头无法找到时，将它复原

}

}

}

return false;

}

## [17. 电话号码的字母组合](https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/)

### 1 原题目

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。



**示例:**

输入："23"

输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

**说明:**

尽管上面的答案是按字典序排列的，但是你可以任意选择答案输出的顺序。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

将字母与数字对应起来，利用回溯加一点暴力搜索的思想来解决问题，具体请看代码和注释。

本人代码：

/\*\*

\* Note: The returned array must be malloced, assume caller calls free().

\*/

int dlen,row;//dlen是记录所给的数字字符数组的长度，row记录的是最后返回数组的行

//digits是题目所给的数组，arr是我们自己开辟的记录结果的数组，n是记录digits数组遍历的位置，col记录arr数组存储到的列数

void dfs(char \*digits,char \*\*arr,int n,int col) {

if(n==dlen) {//当digits数组遍历结束时

arr[row][col]='\0';

row=row+1;//行数加一，开始下一行的存储

int k;

for(k=0;k<=dlen;k++)//此行先保存下上一行的结果，因为回溯借用了上一次的结果

arr[row][k]=arr[row-1][k];

return ;

}

if(digits[n]=='2') {//数字为2时，将所对应的字母依次存进arr数组中并进行递归调用

arr[row][col]='a';

dfs(digits,arr,n+1,col+1);

arr[row][col]='b';

dfs(digits,arr,n+1,col+1);

arr[row][col]='c';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='3') {

arr[row][col]='d';

dfs(digits,arr,n+1,col+1);

arr[row][col]='e';

dfs(digits,arr,n+1,col+1);

arr[row][col]='f';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='4') {

arr[row][col]='g';

dfs(digits,arr,n+1,col+1);

arr[row][col]='h';

dfs(digits,arr,n+1,col+1);

arr[row][col]='i';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='5') {

arr[row][col]='j';

dfs(digits,arr,n+1,col+1);

arr[row][col]='k';

dfs(digits,arr,n+1,col+1);

arr[row][col]='l';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='6') {

arr[row][col]='m';

dfs(digits,arr,n+1,col+1);

arr[row][col]='n';

dfs(digits,arr,n+1,col+1);

arr[row][col]='o';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='7') {

arr[row][col]='p';

dfs(digits,arr,n+1,col+1);

arr[row][col]='q';

dfs(digits,arr,n+1,col+1);

arr[row][col]='r';

dfs(digits,arr,n+1,col+1);

arr[row][col]='s';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='8') {

arr[row][col]='t';

dfs(digits,arr,n+1,col+1);

arr[row][col]='u';

dfs(digits,arr,n+1,col+1);

arr[row][col]='v';

dfs(digits,arr,n+1,col+1);

}

if(digits[n]=='9') {

arr[row][col]='w';

dfs(digits,arr,n+1,col+1);

arr[row][col]='x';

dfs(digits,arr,n+1,col+1);

arr[row][col]='y';

dfs(digits,arr,n+1,col+1);

arr[row][col]='z';

dfs(digits,arr,n+1,col+1);

}

return ;

}

char \*\* letterCombinations(char \* digits, int\* returnSize){

int i;

dlen=strlen(digits);

row=0;

char \*\*arr =(char \*\*)malloc(sizeof(char\*)\*20000);//开辟二维数组

for(i=0; i<20000; i++)

arr[i]=(char \*)malloc(sizeof(char)\*(dlen+1));

if(dlen==0) {//当所给数组长度为0时

\*returnSize=0;

return arr;

}

dfs(digits,arr,0,0);//递归调用

\*returnSize=row;

return arr;

}

## [46. 全排列](https://leetcode-cn.com/problems/permutations/)

### 1 原题目

给定一个没有重复数字的序列，返回其所有可能的全排列。

**示例:**

输入: [1,2,3]

输出:

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/permutations>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

首先需要开辟一个一维的标记数组，用它来标记所给的数字中哪些使用过了，哪些没被使用，然后进行搜索和回溯来找到所有的全排列，具体请看代码及注释。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int row;

void dfs(int \*nums,int \*map,int numsSize,int colu,int \*col,int \*\*arr) {

int i;

if(colu==numsSize) {//当所有数字排列完成之后

col[row]=numsSize;

row=row+1;//下一种情况存在下一行

for(i=0;i<numsSize;i++)//将此种情况存进下一行中，因为之后会用到

arr[row][i]=arr[row-1][i];

return ;

}

for(i=0;i<numsSize;i++) {//遍历题目所给的一维数组

if(map[i]==0) {//找到没被标记的数字

arr[row][colu]=nums[i];//存进二维数组中

map[i]=1;//将此数字标记

dfs(nums,map,numsSize,colu+1,col,arr);//递归调用

map[i]=0;//递归调用回退出来时，要将此数字的标记清空

}

}

return ;

}

int\*\* permute(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i,colu=0;//colu表示即将存进二维数组的哪一列

row=0;//全局变量，表示即将存进二维数组的哪一行

int \*col=(int \*)malloc(50000\*sizeof(int));

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*50000);//开辟二维数组

for(i=0; i<50000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*100);

int \*map=(int \*)malloc(numsSize\*sizeof(int));//开辟一个一维的标记数组

memset(map,0,numsSize\*sizeof(int));//数组初始化

dfs(nums,map,numsSize,colu,col,arr);

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [47. 全排列 II](https://leetcode-cn.com/problems/permutations-ii/)

### 1 原题目

给定一个可包含重复数字的序列，返回所有不重复的全排列。

**示例:**

输入: [1,1,2]

输出:

[

[1,1,2],

[1,2,1],

[2,1,1]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/permutations-ii>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

首先需要开辟一个一维的标记数组，用它来标记所给的数字中哪些使用过了，哪些没被使用，然后进行搜索和回溯来找到所有的全排列并且通过比较去掉重复的序列，具体请看代码及注释。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int row;

void dfs(int \*nums,int \*map,int numsSize,int colu,int \*col,int \*\*arr) {

int i;

if(colu==numsSize) {//当所有数字排列完成之后

if(row==0) {//如果是第0行

col[row]=numsSize;

row=row+1;////下一种情况存在下一行

for(i=0;i<numsSize;i++)//将此种情况存进下一行中，因为之后会用到

arr[row][i]=arr[row-1][i];

}

if(row>0) {//如果是第0行之后的行

int flag=0,j,k;

k=row;

for(i=row-1;i>=0;i--) {//通过比较来判断是否有重复序列

for(j=0;j<numsSize;j++) {

if(arr[k][j]!=arr[i][j])

break;

if(arr[k][j]==arr[i][j]) {

if(j==numsSize-1) {//如果出现有和之前相同的排列

flag=1;//flag标记为1

break;

}

}

}

if(flag==1)//flag为1即为找到重复

break;//跳出循环

}

if(flag==0) {//当没找到重复序列时

col[row]=numsSize;

row=row+1;//行加1，以便下次的存储

for(i=0;i<numsSize;i++)//将此种情况存进下一行中，因为之后会用到

arr[row][i]=arr[row-1][i];

}

}

return ;

}

for(i=0;i<numsSize;i++) {//遍历题目所给的一维数组

if(map[i]==0) {//找到没被标记的数字

arr[row][colu]=nums[i];//存进二维数组中

map[i]=1;//将此数字标记

dfs(nums,map,numsSize,colu+1,col,arr);//递归调用

map[i]=0;//递归调用回退出来时，要将此数字的标记清空

}

}

}

int\*\* permuteUnique(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i,colu=0;//colu表示即将存进二维数组的哪一列

row=0;//全局变量，表示即将存进二维数组的哪一行

int \*col=(int \*)malloc(5000\*sizeof(int));

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*5000);//开辟二维数组

for(i=0; i<5000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*100);

int \*map=(int \*)malloc(numsSize\*sizeof(int));//开辟一个一维的标记数组

memset(map,0,numsSize\*sizeof(int));//数组初始化

dfs(nums,map,numsSize,colu,col,arr);

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [60. 第k个排列](https://leetcode-cn.com/problems/permutation-sequence/)

### 1 原题目

给出集合 [1,2,3,…,n]，其所有元素共有 n! 种排列。

按大小顺序列出所有排列情况，并一一标记，当 n = 3 时, 所有排列如下：

"123"

"132"

"213"

"231"

"312"

"321"

给定 n 和 k，返回第 k 个排列。

**说明：**

给定 n 的范围是 [1, 9]。

给定 k 的范围是[1,  n!]。

**示例 1:**

输入: n = 3, k = 3

输出: "213"

**示例 2:**

输入: n = 4, k = 9

输出: "2314"

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/permutation-sequence>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

此题类似于LeetCode 46题，46题是让你返回所有的全排列，这道题不过是让你返回按照顺序的第K个集合。具体请看代码和注释。

本人代码：

int sum;

void dfs(int \*nums,char \*per,int \*temp,int \*map,int m,int numsSize) {

if(m==numsSize) {//当所有数字排列完成之后

sum=sum-1;//排列的个数减1

if(sum==0) {//当找到第K个排列时

for(int k=0;k<numsSize;k++)

per[k]=temp[k]+'0';//将临时存储元素数组中的元素存到结果数组中

}

return ;

}

int i;

for(i=0;i<numsSize;i++) {//遍历题目所给的一维数组

if(map[i]==0){//找到没被标记的数字

map[i]=1;//将它标记

temp[m]=nums[i];//将它存进临时存储元素的数组中

dfs(nums,per,temp,map,m+1,numsSize);

map[i]=0;//递归调用回退出来时，要将此数字的标记清空

}

}

return ;

}

char \* getPermutation(int n, int k){

int nums[12],i,j=0,temp[n],map[12]={0};//map是标记数组，temp是临时记录元素的数组,nums是存储元素的数组

sum=k;

char \*per=(char \*)malloc((n+1)\*sizeof(char));//记录结果的数组

for(i=1;i<=n;i++)//存储元素进nums数组中

nums[j++]=i;

dfs(nums,per,temp,map,0, n);

per[n]='\0';//结果数组最后要加一个结束符

return per;

}

## [78. 子集](https://leetcode-cn.com/problems/subsets/)

### 1 原题目

给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

**说明：**解集不能包含重复的子集。

**示例:**

输入: nums = [1,2,3]

输出:

[

[3],

  [1],

  [2],

  [1,2,3],

  [1,3],

  [2,3],

  [1,2],

  []

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/subsets>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

这道题参考的代码链接：

<https://leetcode-cn.com/problems/subsets/solution/chui-su-suan-fa-zhi-xing-yong-shi-4-ms-zai-subsets/>

不太会讲，讲不明白，自行看代码和注释理解吧。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int row;

//i是子集中元素的个数，m记录temp数组元素存储的位置，num记录nums数组中遍历到的元素的位置

void dfs(int \*\*arr,int \*nums,int \*temp,int \*col,int i,int m,int num,int numsSize) {

if(i==0) {//当个数为0时

col[row]=m;//记录此行元素的个数

for(int j=0;j<m;j++)//将元素放进arr数组中

arr[row][j]=temp[j];

row=row+1;//开始记录下一行

return ;

}

for(;num<numsSize;num++) {

temp[m]=nums[num];//存进临时数组中

dfs(arr,nums,temp,col,i-1,m+1,num+1,numsSize);

}

return ;

}

int\*\* subsets(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i;

row=1;//从第一行开始存储

int n=(int)pow(2,numsSize);//计算子集的个数

int \*temp=(int \*)malloc(numsSize\*sizeof(int));//临时存放数组中的元素

int \*col=(int \*)malloc(n\*sizeof(int));//开辟一个记录每行有几个元素的数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*n);//开辟二维数组

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

arr[i]=(int \*)malloc(sizeof(int)\*1000);

col[0]=0;//子集之一是空集合

for(i=1;i<=numsSize;i++) {//i为子集中元素的个数

dfs(arr,nums,temp,col,i,0,0,numsSize);

}

\*returnSize=n;

\*returnColumnSizes=col;

return arr;

}

## [90. 子集 II](https://leetcode-cn.com/problems/subsets-ii/)

### 1 原题目

给定一个可能包含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

**说明：**解集不能包含重复的子集。

**示例:**

输入: [1,2,2]

输出:

[

[2],

[1],

[1,2,2],

[2,2],

[1,2],

[]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/subsets-ii>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

不太会讲，讲不明白，自行看代码和注释理解吧（这道题就是LeetCode78题的答案加上一个去重和排序初始数组就可以了）

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int cmp(const void \*a,const void \*b)//比较器

{

return \*(int \*)a-\*(int \*)b;//从小到大排序

}

int row;

//i是子集中元素的个数，m记录temp数组元素存储的位置，num记录nums数组中遍历到的元素的位置

void dfs(int \*\*arr,int \*nums,int \*temp,int \*col,int i,int m,int num,int numsSize) {

if(i==0) {//当个数为0时

int flag;//用来标记是否重复

if(row==1) {//第一行要存进去

col[row]=m;//记录此行元素的个数

for(int j=0;j<m;j++)//将元素放进arr数组中

arr[row][j]=temp[j];

row=row+1;//开始记录下一行

}

else{//从第二行开始要与之前的所有行比较判断是否重复

for(int u=row-1;u>=1;u--) {

flag=0;

if(col[u]!=m) {

flag=1;

continue;

}

for(int v=0;v<col[u];v++)

if(arr[u][v]!=temp[v]) {

flag=1;

break;

}

if(flag==1)

continue;

else

break;

}

}

if(flag==1) {//无重复时

col[row]=m;//记录此行元素的个数

for(int j=0;j<m;j++)//将元素放进arr数组中

arr[row][j]=temp[j];

row=row+1;//开始记录下一行

}

return ;

}

for(;num<numsSize;num++) {

temp[m]=nums[num];//存进临时数组中

i=i-1;//元素个数减1

dfs(arr,nums,temp,col,i,m+1,num+1,numsSize);

i=i+1;//从dfs中跳出就要将元素个数重新加1

}

return ;

}

int\*\* subsetsWithDup(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i;

row=1;//从第一行开始存储

qsort(nums,numsSize,sizeof(nums[0]),cmp);//先将nums数组排序

int n=(int)pow(2,numsSize);//计算子集个数（因为有重复子集所以实际的子集个数比这个少）

int \*temp=(int \*)malloc(numsSize\*sizeof(int));//临时存放数组中的元素

int \*col=(int \*)malloc(n\*sizeof(int));//开辟一个记录每行有几个元素的数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*n);//开辟二维数组

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

arr[i]=(int \*)malloc(sizeof(int)\*1000);

col[0]=0;//子集之一是空集合

for(i=1;i<=numsSize;i++) {//i为子集中元素的个数

dfs(arr,nums,temp,col,i,0,0,numsSize);

}

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [77. 组合](https://leetcode-cn.com/problems/combinations/)

### 1 原题目

给定两个整数 n 和 k，返回 1 ... n 中所有可能的 k 个数的组合。

**示例:**

输入: n = 4, k = 2

输出:

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/combinations>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

返回1—n数字中含K个元素的子集，（就是LeetCode 78题中所给的那个数组我们手动创建而已），具体大家请看代码和注释。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int row;

//i是子集中元素的个数，m记录temp数组元素存储的位置，num记录nums数组中遍历到的元素的位置

void dfs(int \*\*arr,int \*nums,int \*temp,int \*col,int i,int m,int num,int numsSize) {

if(i==0) {//当个数为0时

col[row]=m;//记录此行元素的个数

for(int i=0;i<m;i++)//将元素放进arr数组中

arr[row][i]=temp[i];

row=row+1;//开始记录下一行

return ;

}

for(;num<numsSize;num++) {

temp[m]=nums[num];//存进临时数组中

i=i-1;//元素个数减1

dfs(arr,nums,temp,col,i,m+1,num+1,numsSize);

i=i+1;//从dfs中跳出就要将元素个数重新加1

}

return ;

}

int\*\* combine(int n, int k, int\* returnSize, int\*\* returnColumnSizes){

int nums[1000]={0},numsSize=0,i;//num数组存储题意所说的1—n的数字

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

nums[numsSize++]=i;

row=0;

int temp[numsSize];//临时存放数组中的元素

int \*col=(int \*)malloc(10000\*sizeof(int));//开辟一个记录每行有几个元素的数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*10000);//开辟二维数组

for(i=0; i<10000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*k);

dfs(arr,nums,temp,col,k,0,0,numsSize);

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [39. 组合总和](https://leetcode-cn.com/problems/combination-sum/)

### 1 原题目

给定一个无重复元素的数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的数字可以无限制重复被选取。

**说明：**

所有数字（包括 target）都是正整数。

解集不能包含重复的组合。

**示例 1:**

输入: candidates = [2,3,6,7], target = 7,

所求解集为:

[

[7],

[2,2,3]

]

**示例 2:**

输入: candidates = [2,3,5], target = 8,

所求解集为:

[

  [2,2,2,2],

  [2,3,3],

  [3,5]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/combination-sum/>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

此题类似于LeetCode 40题（这道题因为初始数组无重复元素，所以不需要像40题那样去重和排序，但是这道题因为数组中的元素可以重复使用，所以就是在dfs进行递归的时候记录数组遍历到哪里的num不需要加一），具体请看代码和注释。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int row,n;

//sum是已加到临时数组中的元素的总和，m记录temp数组元素存储的位置，num记录candidates数组中遍历到的元素的位置

void dfs(int \*\*arr,int \*candidates,int \*temp,int \*col,int sum,int m,int num,int candidatesSize) {

if(sum==n) {//当和为题目所给值时

col[row]=m;//记录此行元素的个数

for(int j=0;j<m;j++)//将元素放进arr数组中

arr[row][j]=temp[j];

row=row+1;//开始记录下一行

return ;

}

for(;num<candidatesSize;num++) {

if(candidates[num]>n-sum)//判断当前数字是否可以加到临时数组中

continue;

temp[m]=candidates[num];//存进临时数组中

sum=sum+candidates[num];//总和加上当前元素

dfs(arr,candidates,temp,col,sum,m+1,num,candidatesSize);//因为数组中的元素可以重复使用，num不用加1

sum=sum-candidates[num];//从dfs中跳出时，要减去当前元素

}

return ;

}

int\*\* combinationSum(int\* candidates, int candidatesSize, int target, int\* returnSize, int\*\* returnColumnSizes){

n=target;//n是全局变量，记录所给值

row=0;

int sum=0,temp[100],i;//temp是临时存元素的数组

int \*col=(int \*)malloc(1000\*sizeof(int));//开辟一个记录每行有几个元素的数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*1000);//开辟二维数组

for(i=0; i<1000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*100);

dfs(arr,candidates,temp,col,sum,0,0,candidatesSize);

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [40. 组合总和 II](https://leetcode-cn.com/problems/combination-sum-ii/)

### 1 原题目

给定一个数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的每个数字在每个组合中只能使用一次。

**说明：**

所有数字（包括目标数）都是正整数。

解集不能包含重复的组合。

**示例 1:**

输入: candidates = [10,1,2,7,6,1,5], target = 8,

所求解集为:

[

[1, 7],

[1, 2, 5],

[2, 6],

[1, 1, 6]

]

**示例 2:**

输入: candidates = [2,5,2,1,2], target = 5,

所求解集为:

[

  [1,2,2],

  [5]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/combination-sum-ii>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

此题类似于LeetCode 90题（90题是有重复元素找出所有不同的子集，此题不过是找出元素之和等于所给值而已），具体请看代码和注释。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int cmp(const void \*a,const void \*b) {//比较器

return \*(int \*)a-\*(int \*)b;//从小到大排序

}

int row,n;

//sum是已加到临时数组中的元素的总和，m记录temp数组元素存储的位置，num记录candidates数组中遍历到的元素的位置

void dfs(int \*\*arr,int \*candidates,int \*temp,int \*col,int sum,int m,int num,int candidatesSize) {

if(sum==n) {//当和为题目所给值时

int flag;

if(row==0) {//第0行一定要先存储的

col[row]=m;//记录此行元素的个数

for(int j=0;j<m;j++)//将元素放进arr数组中

arr[row][j]=temp[j];

row=row+1;//开始记录下一行

}

else{//从第一行开始要与之前的所有行比较判断是否重复

for(int u=row-1;u>=0;u--) {

flag=0;

if(col[u]!=m) {

flag=1;

continue;

}

for(int v=0;v<col[u];v++)

if(arr[u][v]!=temp[v]) {

flag=1;

break;

}

if(flag==1)

continue;

else

break;

}

}

if(flag==1) {//无重复时

col[row]=m;//记录此行元素的个数

for(int j=0;j<m;j++)//将元素放进arr数组中

arr[row][j]=temp[j];

row=row+1;//开始记录下一行

}

return ;

}

for(;num<candidatesSize;num++) {

if(candidates[num]>n-sum)//判断当前数字是否可以加到临时数组中

continue;

temp[m]=candidates[num];//存进临时数组中

sum=sum+candidates[num];//总和加上当前元素

dfs(arr,candidates,temp,col,sum,m+1,num+1,candidatesSize);

sum=sum-candidates[num];//从dfs中跳出时，要减去当前元素

}

return ;

}

int\*\* combinationSum2(int\* candidates, int candidatesSize, int target, int\* returnSize, int\*\* returnColumnSizes){

qsort(candidates,candidatesSize,sizeof(candidates[0]),cmp);//排序

n=target;//n是全局变量，记录所给值

row=0;

int sum=0,temp[candidatesSize],i;//temp是临时存元素的数组

int \*col=(int \*)malloc(1000\*sizeof(int));//开辟一个记录每行有几个元素的数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*1000);//开辟二维数组

for(i=0; i<1000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*candidatesSize);

dfs(arr,candidates,temp,col,sum,0,0,candidatesSize);

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [216. 组合总和 III](https://leetcode-cn.com/problems/combination-sum-iii/)

### 1 原题目

找出所有相加之和为 n 的 k 个数的组合。组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

**说明：**

所有数字都是正整数。

解集不能包含重复的组合。

**示例 1:**

输入: k = 3, n = 7

输出: [[1,2,4]]

**示例 2:**

输入: k = 3, n = 9

输出: [[1,2,6], [1,3,5], [2,3,4]]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/combination-sum-iii>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

回溯算法（解法一用到）

### 3 解法

#### 3.1 解法1（C语言已过）

回溯算法的介绍及讲解：

<https://time.geekbang.org/column/article/74287?register=0&gk_activity=0>

返回1—9数字中和为n的含K个元素的子集，（就是LeetCode 78题中所给的那个数组我们手动创建而已再加上一个判断元素和是否等于所给值），具体大家请看代码和注释。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int row,n1;

//sum是已加到临时数组中的元素的总和，i是子集中元素的个数，m记录temp数组元素存储的位置，num记录nums数组中遍历到的元素的位置

void dfs(int \*\*arr,int \*nums,int \*temp,int \*col,int sum,int i,int m,int num,int numsSize) {

if(i==0&&sum==n1) {//当元素个数为0并且和为所给值

col[row]=m;//记录此行元素的个数

for(int i=0;i<m;i++)//将元素放进arr数组中

arr[row][i]=temp[i];

row=row+1;//开始记录下一行

return ;

}

for(;num<numsSize;num++) {

if(nums[num]>n1-sum)//判断当前数字是否可以加到临时数组中

continue;

temp[m]=nums[num];//存进临时数组

i=i-1;//个数减一

sum=sum+nums[num];//总和加上当前元素

dfs(arr,nums,temp,col,sum,i,m+1,num+1,numsSize);

i=i+1;//从dfs中跳出时，个数加一

sum=sum-nums[num];//从dfs中跳出时，要减去当前元素

}

return ;

}

int\*\* combinationSum3(int k, int n, int\* returnSize, int\*\* returnColumnSizes){

int nums[10],i,numsSize=0,sum=0;//nums数组存储1—9这9个数

n1=n;//n1是全局变量，记录所给值n

for(i=1;i<=9;i++)

nums[numsSize++]=i;

row=0;

int temp[numsSize];//临时存放数组中的元素

int \*col=(int \*)malloc(1000\*sizeof(int));//开辟一个记录每行有几个元素的数组

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*1000);//开辟二维数组

for(i=0; i<1000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*k);

dfs(arr,nums,temp,col,sum,k,0,0,numsSize);

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [31. 下一个排列](https://leetcode-cn.com/problems/next-permutation/)

### 1 原题目

实现获取下一个排列的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须原地修改，只允许使用额外常数空间。

以下是一些例子，输入位于左侧列，其相应输出位于右侧列。

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/next-permutation>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

从后往前扫描数组找到第一个下标i,使得nums[i] < nums[i + 1]；若i = -1，表示该数组为单调递减序列;否则，再次从后往前扫描数组找到找到第一个下标j使得nums[j] > nums[i]，交换两数，最后翻转i+1到numsSize-1这一段的元素

本人代码：

void nextPermutation(int\* nums, int numsSize){

int j,i,temp,num=numsSize;

if(numsSize==0)//数组大小为0时

return ;

for(i=numsSize-2;i>=0;i--) {//找到第一个i使nums[i]<nums[i+1]

if(nums[i+1]>nums[i])

break;

}

if(i>=0) {//如果此时数组没遍历完

for(j=numsSize-1;j>i;j--) {//找到第一个j使nums[j]>nums[i]

if(nums[j]>nums[i]) {//交换nums[i]和nums[j]

temp=nums[j];

nums[j]=nums[i];

nums[i]=temp;

break;

}

}

}

for(j=i+1;j<(i+1+numsSize)/2;j++) {//翻转i+1到numsSize-1这一段的元素

temp=nums[j];

nums[j]=nums[num-1];

nums[num-1]=temp;

num--;

}

}

## [15. 三数之和](https://leetcode-cn.com/problems/3sum/)

### 1 原题目

给定一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？找出所有满足条件且不重复的三元组。

**注意：**答案中不可以包含重复的三元组。

例如, 给定数组 nums = [-1, 0, 1, 2, -1, -4]，

满足要求的三元组集合为：

[

[-1, 0, 1],

[-1, -1, 2]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/3sum>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将数组中的元素从小到大排好序，然后利用双指针来解决问题。先遍历数组，当遍历第一个元素时，左指针指向它后一个元素，右指针指向最后一个元素，此三个元素相加，大于零右指针前移，小于零左指针后移。以此类推。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int cmp(const void \*a,const void \*b)//比较器

{

return \*(int \*)a-\*(int \*)b; //从小到大排序

}

int\*\* threeSum(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i,left,right,row=0,s;

int \*col=(int \*)malloc(17000\*sizeof(int));

qsort(nums,numsSize,sizeof(nums[0]),cmp);//排序函数

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*17000);//arr为二维数组

for(i=0; i<17000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*3);

for(i=0;i<numsSize-2;i++) {//遍历数组

if(i>0&&nums[i]==nums[i-1])//去重

continue;

left=i+1;//左指针

right=numsSize-1;//右指针

while(left<right) {//循环的条件

s=nums[i]+nums[left]+nums[right];

if(s<0)

left++;

if(s>0)

right--;

if(s==0) {//找到答案

arr[row][0]=nums[i];

arr[row][1]=nums[left];

arr[row][2]=nums[right];

col[row]=3;

row++;

while(left<right&&nums[left]==nums[left+1])//去重

left++;

while(left<right&&nums[right]==nums[right-1])//去重

right--;

left++;

right--;

}

}

}

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [18. 四数之和](https://leetcode-cn.com/problems/4sum/)

### 1 原题目

给定一个包含 n 个整数的数组 nums 和一个目标值 target，判断 nums 中是否存在四个元素 a，b，c 和 d ，使得 a + b + c + d 的值与 target 相等？找出所有满足条件且不重复的四元组。

**注意：**

答案中不可以包含重复的四元组。

**示例：**

给定数组 nums = [1, 0, -1, 0, -2, 2]，和 target = 0。

满足要求的四元组集合为：

[

[-1, 0, 0, 1],

[-2, -1, 1, 2],

[-2, 0, 0, 2]

]

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/4sum>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将数组中的元素从小到大排好序，然后利用双指针来解决问题。用双层的for循环嵌套遍历数组，当第一层遍历第一个元素时，第二层遍历第二个元素时，左指针指向第二层元素后一个元素，右指针指向最后一个元素，此四个元素相加，大于零右指针前移，小于零左指针后移。以此类推。

本人代码：

/\*\*

\* Return an array of arrays of size \*returnSize.

\* The sizes of the arrays are returned as \*returnColumnSizes array.

\* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

\*/

int cmp(const void \*a,const void \*b)//比较器

{

return \*(int \*)a-\*(int \*)b;

}

int\*\* fourSum(int\* nums, int numsSize, int target, int\* returnSize, int\*\* returnColumnSizes){

qsort(nums,numsSize,sizeof(nums[0]),cmp);//排序函数

int i,j,row=0,s,left,right,flag=0;

int \*col=(int \*)malloc(1000\*sizeof(int));

int \*\*arr =(int \*\*)malloc(sizeof(int\*)\*1000);//开辟二维数组

for(i=0; i<1000; i++)

arr[i]=(int \*)malloc(sizeof(int)\*4);

for(i=0;i<numsSize-3;i++) {//遍历数组

for(j=i+1;j<numsSize-2;j++) {

if(j>i+1&&nums[j]==nums[j-1]||(i>0&&nums[i]==nums[i-1]))//去重

continue;

left=j+1;//左指针

right=numsSize-1;//右指针

while(left<right) {//循环条件

s=nums[i]+nums[j]+nums[left]+nums[right];

if(s<target)

left++;

if(s>target)

right--;

if(s==target) {

arr[row][0]=nums[i];

arr[row][1]=nums[j];

arr[row][2]=nums[left];

arr[row][3]=nums[right];

col[row]=4;

row++;

while(left<right&&nums[left]==nums[left+1])//去重

left++;

while(left<right&&nums[right]==nums[right-1])//去重

right--;

left++;

right--;

}

}

}

}

\*returnSize=row;

\*returnColumnSizes=col;

return arr;

}

## [29. 两数相除](https://leetcode-cn.com/problems/divide-two-integers/)

### 1 原题目

给定两个整数，被除数 dividend 和除数 divisor。将两数相除，要求不使用乘法、除法和 mod 运算符。

返回被除数 dividend 除以除数 divisor 得到的商。

**示例 1:**

输入: dividend = 10, divisor = 3

输出: 3

**示例 2:**

输入: dividend = 7, divisor = -3

输出: -2

**说明:**

被除数和除数均为 32 位有符号整数。

除数不为 0。

假设我们的环境只能存储 32 位有符号整数，其数值范围是 [−231,  231 − 1]。本题中，如果除法结果溢出，则返回 231 − 1。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/divide-two-integers>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

计算除法的结果其实就是计算出来几个除数加起来等于被除数，利用这一点就可以写出这道题（注意除数不能一个个的加，最差要一次加三个）

本人代码：

int divide(int dividend, int divisor){

if(dividend==divisor)//除数和被除数相等时

return 1;

if(dividend==INT\_MIN) {//当被除数为负数最小值时

if(divisor==1)//除数为1时

return dividend;

if(divisor==-1)//除数为-1时

return INT\_MAX;

}

//当除数为负数最小值时或者被除数的绝对值小于除数的绝对值

if(divisor==INT\_MIN||labs(dividend)<labs(divisor))

return 0;

if(divisor==1){//当除数为1时

return dividend;

}

int sum=0,i=0;//sum记录除数相加的和，i记录除数相加的数量

int div=dividend,iso=divisor;

if(div>0&&iso>0){//被除数和除数都大于0时

while(iso<=429496729&&5\*iso<=div-sum) {//如果可以先五次五次的加

sum=sum+5\*iso;

i=i+5;

}

while(iso<=div-sum) {//再一次一次的加

sum=sum+iso;

i=i+1;

}

}

if(div<0&&iso<0) {//被除数和除数都小于0时

while(iso>=-429496729&&5\*iso>=div-sum) {

sum=sum+5\*iso;

i=i+5;

}

while(iso>=div-sum) {

sum=sum+iso;

i=i+1;

}

}

if(div>0&&iso<0) {//被除数大于0，除数小于0时

div=-div;//因为负数的最小值的绝对值比正数的最大值大，所以换成负数

while(iso>=-429496729&&5\*iso>=div-sum) {

sum=sum+5\*iso;

i=i+5;

}

while(iso>=div-sum) {

sum=sum+iso;

i=i+1;

}

i=-i;//结果是负数

}

if(div<0&&iso>0) {//被除数小于0，除数大于0时

iso=-iso;

while(iso>=-429496729&&5\*iso>=div-sum) {

sum=sum+5\*iso;

i=i+5;

}

while(iso>=div-sum) {

sum=sum+iso;

i=i+1;

}

i=-i;

}

return i;

}

## [150. 逆波兰表达式求值](https://leetcode-cn.com/problems/evaluate-reverse-polish-notation/)

### 1 原题目

根据逆波兰表示法，求表达式的值。

有效的运算符包括 +, -, \*, / 。每个运算对象可以是整数，也可以是另一个逆波兰表达式。

**说明：**

整数除法只保留整数部分。

给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。

**示例 1：**

输入: ["2", "1", "+", "3", "\*"]

输出: 9

解释: ((2 + 1) \* 3) = 9

**示例 2：**

输入: ["4", "13", "5", "/", "+"]

输出: 6

解释: (4 + (13 / 5)) = 6

**示例 3：**

输入: ["10", "6", "9", "3", "+", "-11", "\*", "/", "\*", "17", "+", "5", "+"]

输出: 22

解释:

((10 \* (6 / ((9 + 3) \* -11))) + 17) + 5

= ((10 \* (6 / (12 \* -11))) + 17) + 5

= ((10 \* (6 / -132)) + 17) + 5

= ((10 \* 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/evaluate-reverse-polish-notation>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

利用整型栈来做题，遍历字符数组将数字存进栈中，如果遍历到运算符，就从栈顶选两个数字进行运算，然后在将运算结果存进栈中，以此类推直到字符数组遍历完。

本人代码：

int evalRPN(char \*\* tokens, int tokensSize){

    long long int arr[10000] = {0};//存放数字的栈

    int top = 0;//栈的指针

    for(int i=0;i<tokensSize;i++) {//遍历题目所给的二维数组

        if(strlen(tokens[i]) == 1) {//当前行长度为一时

            if(tokens[i][0] >= '0' && tokens[i][0] <= '9')//如果是一位数字就存进栈中

                arr[top++] = tokens[i][0] - '0';

            else {//长度只有一位不是一位数字就是运算符

                switch(tokens[i][0])//判断是哪个运算符

                {

                    case '+':

                        arr[top-2] = arr[top-2] + arr[top-1];

                        top = top - 1;

                        break;

                    case '-':

                        arr[top-2] = arr[top-2] - arr[top-1];

                        top = top - 1;

                        break;

                    case '\*':

                        arr[top-2] = arr[top-2] \* arr[top-1];

                        top = top - 1;

                        break;

                    case '/':

                        arr[top-2] = arr[top-2] / arr[top-1];

                        top = top - 1;

                        break;

                }

            }

        }

        else {//当前行长度大于一时

            int sum = 0,num = 0;

            if(tokens[i][0] != '-') {//不是负数的情况

                for(int j=strlen(tokens[i])-1;j>=0;j--) {

                    sum = sum + (int)pow(10,num++) \* (tokens[i][j] - '0');//将字符的数字计算成整型的数字

                }

                arr[top++] = sum;

            }

            else {//是负数的情况

                for(int j=strlen(tokens[i])-1;j>0;j--) {

                    sum = sum + (int)pow(10,num++) \* (tokens[i][j] - '0');

                }

                arr[top++] = -sum;

            }

        }

    }

    return arr[0];

}

## [227. 基本计算器 II](https://leetcode-cn.com/problems/basic-calculator-ii/)

### 1 原题目

实现一个基本的计算器来计算一个简单的字符串表达式的值。

字符串表达式仅包含非负整数，+， - ，\*，/ 四种运算符和空格  。 整数除法仅保留整数部分。

**示例 1:**

输入: "3+2\*2"

输出: 7

**示例 2:**

输入: " 3/2 "

输出: 1

**示例 3:**

输入: " 3+5 / 2 "

输出: 5

**说明：**

你可以假设所给定的表达式都是有效的。

请不要使用内置的库函数 eval。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/basic-calculator-ii>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

因为题意告诉我们所给的数字都是非负整数，所以我解这道题地思路就是先将用字符串表达的式子用整型数组存储，数字部分好说，加减乘除符号我们可以用负数来表示（本人用的是-1到-4），然后就照常的进行先乘除后加减的运算。

本人代码：

int arr[200000],arr1[200000];

int calculate(char \* s){

int slen=strlen(s),i,num=0,sum,num1=0,bit;//bit是记录位数

memset(arr,0,sizeof(arr));//初始化数组

memset(arr1,0,sizeof(arr1));//初始化数组

for(i=0;i<slen;i++) {//将符号用数字记录

sum = 0;

bit = 0;

if(s[i] == ' ')

continue;

if(s[i] == '+')

arr[num++] = -1;

if(s[i] == '-')

arr[num++] = -2;

if(s[i] == '\*')

arr[num++] = -3;

if(s[i] == '/')

arr[num++] = -4;

if(s[i] >= '0' && s[i] <= '9') {//将多位数转成整型存起来

int temp = i;

while(i <= slen - 1 && s[i] >= '0' && s[i] <= '9')

i = i + 1;

for(int j=i-1;j>=temp;j--)

sum = sum + pow(10,bit++) \* (s[j] - '0');

arr[num++] = sum;

i = i -1;

}

}

for(i=0;i<num;i++) {//先将arr数组中乘法和除法计算出来存储，其余原样存储

if(arr[i] != -3 && arr[i] != -4)

arr1[num1++] = arr[i];

if(arr[i] == -3) {

int cf = arr[i-1];

do{

if(arr[i] == -3){

cf = cf \* arr[i+1];

i = i + 2;

}

if(arr[i] == -4){

cf = cf / arr[i+1];

i = i + 2;

}

}while(arr[i] == -3 || arr[i] == -4);

arr1[num1-1] = cf;

i = i - 1;

continue;

}

if(arr[i] == -4) {

int cf = arr[i-1];

do{

if(arr[i] == -3){

cf = cf \* arr[i+1];

i = i + 2;

}

if(arr[i] == -4){

cf = cf / arr[i+1];

i = i + 2;

}

}while(arr[i] == -3 || arr[i] == -4);

arr1[num1-1] = cf;

i = i - 1;

continue;

}

}

sum = 0;

sum = sum + arr1[0];

for(i=1;i<num1-1;i=i+2) {//计算加减运算

if(arr1[i] == -1)

sum = sum + arr1[i+1];

if(arr1[i] == -2)

sum = sum - arr1[i+1];

}

return sum;

}

## [56. 合并区间](https://leetcode-cn.com/problems/merge-intervals/)

### 1 原题目

给出一个区间的集合，请合并所有重叠的区间。

**示例 1:**

输入: [[1,3],[2,6],[8,10],[15,18]]

输出: [[1,6],[8,10],[15,18]]

解释: 区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].

**示例 2:**

输入: [[1,4],[4,5]]

输出: [[1,5]]

解释: 区间 [1,4] 和 [4,5] 可被视为重叠区间。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/merge-intervals>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将传进来的多个区间按照左边界的大小将他排序成由小到大，这样可以合并的区间必然是连续的。然后进行区间的比较将区间合并。

本人代码：

int cmp(const void \*p1, const void \*p2){//比较器

    return (\*((int \*\*)p1))[0] - (\*((int \*\*)p2))[0];

}

int\*\* merge(int\*\* intervals, int intervalsSize, int\* intervalsColSize, int\* returnSize, int\*\* returnColumnSizes){

qsort(intervals,intervalsSize, sizeof(intervals[0]),cmp);//将二维数组按照左边界的大小来进行从小到大排序

int i,row = 0;

int \*\*arr = (int \*\*)malloc(sizeof(int \*)\*1000);//开辟二维数组

for(i=0;i<1000;i++)

arr[i] = (int \*)malloc(sizeof(int)\*2);

for(i=0;i<intervalsSize;i++) {

arr[row][0] = intervals[i][0];

arr[row][1] = intervals[i][1];

while(i<intervalsSize-1 && arr[row][1]>=intervals[i+1][0]) {//如果后一行的左边界小于等于前一行的右边界就说明可以合并

arr[row][1] = arr[row][1]>intervals[i+1][1]?arr[row][1]:intervals[i+1][1];//合并后的区间的右边界要选择大的那一个

i = i + 1;

}

row = row + 1;

}

\*returnSize = row;

\*returnColumnSizes = (int \*)malloc(sizeof(int)\*row);

for(i=0;i<row;i++)

(\*returnColumnSizes)[i] = 2;

return arr;

}

## [57. 插入区间](https://leetcode-cn.com/problems/insert-interval/)

### 1 原题目

给出一个无重叠的 ，按照区间起始端点排序的区间列表。

在列表中插入一个新的区间，你需要确保列表中的区间仍然有序且不重叠（如果有必要的话，可以合并区间）。

**示例 1:**

输入: intervals = [[1,3],[6,9]], newInterval = [2,5]

输出: [[1,5],[6,9]]

**示例 2:**

输入: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

输出: [[1,2],[3,10],[12,16]]

解释: 这是因为新的区间 [4,8] 与 [3,5],[6,7],[8,10] 重叠。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/insert-interval>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

因为传进来的多个区间列表按照左边界的大小排序成由小到大，我们先将新区间按顺序直接插到区间列表中，然后进行区间合并就可以了。

本人代码：

int\*\* insert(int\*\* intervals, int intervalsSize, int\* intervalsColSize, int\* newInterval, int newIntervalSize, int\* returnSize, int\*\* returnColumnSizes){

int num[intervalsSize+1][2],i,row = 0,flag = 0;

//将新区间按左边界的大小插入到区间列表中

for(i=0;i<intervalsSize;i++) {//将区间列表的左边界值小于等于新区间左边界的存入num中

if(intervals[i][0]<=newInterval[0]) {

num[row][0] = intervals[i][0];

num[row][1] = intervals[i][1];

row++;

}

else {//新区间的存入

num[row][0] = newInterval[0];

num[row][1] = newInterval[1];

row++;

flag = 1;//如果新区间插入进去了，标记为1

break;//新区间插入进取后就跳出循环

}

}

for(;i<intervalsSize;i++) {//将区间列表左边界值大于新区间的左边界值存入num中（后半部分都是）

num[row][0] = intervals[i][0];

num[row][1] = intervals[i][1];

row++;

}

if(intervalsSize == 0 || flag == 0) {//当所给的区间列表为空时或者新区间未插入进去（flag未被标记）

num[row][0] = newInterval[0];

num[row][1] = newInterval[1];

}

//区间合并

row = 0;

int \*\*arr = (int \*\*)malloc(sizeof(int \*)\*1000);//开辟二维数组

for(i=0;i<1000;i++)

arr[i] = (int \*)malloc(sizeof(int)\*2);

for(i=0;i<intervalsSize+1;i++) {

arr[row][0] = num[i][0];

arr[row][1] = num[i][1];

while(i<intervalsSize && arr[row][1]>=num[i+1][0]) {//如果后一行的左边界小于等于前一行的右边界就说明可以合并

arr[row][1] = arr[row][1]>num[i+1][1]?arr[row][1]:num[i+1][1];//合并后的区间的右边界要选择大的那一个

i = i + 1;

}

row = row + 1;

}

\*returnSize = row;

\*returnColumnSizes = (int \*)malloc(sizeof(int)\*row);

for(i=0;i<row;i++)

(\*returnColumnSizes)[i] = 2;

return arr;

}

## [45. 跳跃游戏 II](https://leetcode-cn.com/problems/jump-game-ii/)

### 1 原题目

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

你的目标是使用最少的跳跃次数到达数组的最后一个位置。

**示例:**

输入: [2,3,1,1,4]

输出: 2

解释: 跳到最后一个位置的最小跳跃数是 2。

  从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。

**说明:**

假设你总是可以到达数组的最后一个位置。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

贪心算法（解法二用到）

### 3 解法

#### 3.1 解法1（C语言已过）

要用最少的步数到达数组最后一个位置，我们可以从数组的尾部开始，找到能到达它的位置并离它最远（从左到右遍历数组，第一个找到的就是）。然后继续向前寻找，最后到了第 0 个位置就结束了。

本人代码：

|  |
| --- |
| int jump(int\* nums, int numsSize){      int pos = numsSize - 1,step = 0; //pos为要找的位置      while (pos > 0) { //是否到了第 0 个位置          for (int i = 0; i < pos; i++) {              if (nums[i] + i >= pos) {                  pos = i; //更新要找的位置                  step++;                  break;              }          }      }      return step;  } |

#### 3.2 解法2（C语言已过）

利用贪心来寻找当前步数跳到哪个位置可以使跳跃距离最远。

i到end之间是当前步，maxpos是在当前步找下一步的最远距离。

在遍历数组时，我们不访问最后一个元素，这是因为在访问最后一个元素之前，我们的边界一定大于等于最后一个位置，否则就无法跳到最后一个位置了。如果访问最后一个元素，在边界正好为最后一个位置的情况下，我们会增加一次不必要的跳跃次数，因此我们不必访问最后一个元素。

参考链接：<https://leetcode-cn.com/problems/jump-game-ii/solution/tiao-yue-you-xi-ii-by-leetcode-solution/>

本人代码：

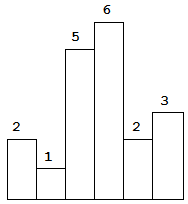
|  |
| --- |
| int jump(int\* nums, int numsSize){      //step步数,end记录当前跳跃次数内的下一步最大下标,maxpos记录i到end(当前步骤)的下一跳的最大长度      int step = 0,end = 0,maxpos = 0;      for(int i = 0; i < numsSize - 1; i++) {          maxpos = maxpos > i + nums[i] ? maxpos : i + nums[i];          if(i == end) {              end = maxpos;              step++;          }      }      return step;  } |

## [84. 柱状图中最大的矩形](https://leetcode-cn.com/problems/largest-rectangle-in-histogram/)

### 1 原题目

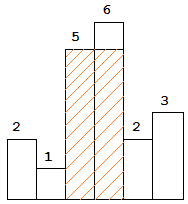
给定 n 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。

求在该柱状图中，能够勾勒出来的矩形的最大面积。



以上是柱状图的示例，其中每个柱子的宽度为 1，给定的高度为 [2,1,5,6,2,3]。

图中阴影部分为所能勾勒出的最大矩形面积，其面积为 10 个单位。



**示例:**

输入: [2,1,5,6,2,3]

输出: 10

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/largest-rectangle-in-histogram>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（java已过）

本人用的是暴力法求解，从第一个柱体尝试与后面的n个柱体结合，会有一个最大面积值，然后继续从第二个柱体尝试，通过不断的比较这样在最后就会求解出一个最大的矩形面积值。

本人代码：

class Solution {

    public int largestRectangleArea(int[] heights) {

        int maxarea = 0, minheight;

        for(int i = 0; i < heights.length; i++) {

            minheight = Integer.MAX\_VALUE;

            for(int j = i; j < heights.length; j++) {//第i个柱形与后面柱形组成的矩形的最大面积

                minheight = Math.min(minheight,heights[j]);//柱形的高由高度最低的那一个柱形决定的

                maxarea = Math.max(maxarea,minheight \* (j - i + 1));

            }

        }

        return maxarea;

    }

}

## [43. 字符串相乘](https://leetcode-cn.com/problems/multiply-strings/)

### 1 原题目

给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2 的乘积，它们的乘积也表示为字符串形式。

**示例 1:**

输入: num1 = "2", num2 = "3"

输出: "6"

**示例 2:**

输入: num1 = "123", num2 = "456"

输出: "56088"

**说明：**

num1 和 num2 的长度小于110。

num1 和 num2 只包含数字 0-9。

num1 和 num2 均不以零开头，除非是数字 0 本身。

不能使用任何标准库的大数类型（比如 BigInteger）或直接将输入转换为整数来处理。

### 2 基础知识

### 3 解法

#### 3.1 解法1（java已过）

具体的思想及解释请看这个链接：<https://mp.weixin.qq.com/s?__biz=MzAxODQxMDM0Mw==&mid=2247484466&idx=1&sn=0281340cc1f41230e4512e905b9d27dd&chksm=9bd7fa3aaca0732c95d25c637d42ad8d9b80f8165098ded837f83791c673b5d6a71721c738a3&scene=21#wechat_redirect>

本人代码：

class Solution {

    public String multiply(String num1, String num2) {

        if(num1.equals("0") || num2.equals("0"))//当出现字符串为"0"的情况

            return "0";

        int num1len = num1.length(), num2len = num2.length(),i,j;

        int res[] = new int[num1len + num2len];//存放整型的字符串相乘后的结果

        for(i = num1len - 1; i >= 0; i--) {//从个位数开始逐位相乘

            for(j = num2len - 1; j >= 0; j--) {

                int mul = (num1.charAt(i) - '0') \* (num2.charAt(j) - '0');

                int p1 = i + j, p2 = i + j + 1;//乘积mul在res数组中的位置

                int sum = mul + res[p2];

                res[p1] = res[p1] + sum / 10;//将sum叠加到res数组中

                res[p2] = sum % 10;

            }

        }

        i = 0;

        while(i < num1len + num2len && res[i] == 0)//res数组前面可能有没用到的（未使用位）

            i++;

        String res1 = new String();//字符串

        StringBuilder builder = new StringBuilder();

        for(int z = i; z < num1len + num2len; z++)//将整型数组res中的值转换成字符串

            builder.append(res[z]);

        return builder.toString();//返回字符串

    }

}

## [49. 字母异位词分组](https://leetcode-cn.com/problems/group-anagrams/)

### 1 原题目

给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。

**示例:**

输入: ["eat", "tea", "tan", "ate", "nat", "bat"],

输出:

[

["ate","eat","tea"],

["nat","tan"],

["bat"]

]

**说明：**

所有输入均为小写字母。

不考虑答案输出的顺序。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/group-anagrams>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

先将题目给的二维字符数组strs中的字符串存到另外的strscpy中，然后将strscpy数组中的每一行的字符串进行排序，排好序后进行遍历比较就可以完成字母异位词放在一个二维数组中。

本人代码：

/\*\*

 \* Return an array of arrays of size \*returnSize.

 \* The sizes of the arrays are returned as \*returnColumnSizes array.

 \* Note: Both returned array and \*columnSizes array must be malloced, assume caller calls free().

 \*/

int cmp(const void \*a,const void \*b)

{

     return(\*(char \*)a-\*(char \*)b);

}

char \*\*\* groupAnagrams(char \*\* strs, int strsSize, int\* returnSize, int\*\* returnColumnSizes){

    \*returnColumnSizes = (int \*)malloc(sizeof(int) \* strsSize);

    int book[strsSize],i,j;//book为标记数组

    char \*\*strscpy = (char \*\*)malloc(sizeof(char \*) \* strsSize);//将strs数组中的字符串赋给strscpy数组

    for(i = 0; i < strsSize; i++) {

        book[i] = 0;//标记数组置0

        strscpy[i] = (char \*)malloc(sizeof(char) \* strlen(strs[i]) + 1);

        strcpy(strscpy[i], strs[i]);

        qsort(strscpy[i],strlen(strscpy[i]),sizeof(strscpy[i][0]),cmp);//将每一行的字符串进行排序

    }

    char \*\*\*res = (char \*\*\*)malloc(sizeof(char \*\*)\*strsSize);//开辟三维数组记录结果

    int row = 0,col = 0;//row记录的是三维数组的层数，col记录三维数组每一层的行数

    for(i = 0; i < strsSize; i++) {//遍历整个二维的stscpy数组

        if(book[i] == 1)//如果当前行被标记就跳过

            continue;

        book[i] = 1;//标记当前行

        res[row] = (char \*\*)malloc(sizeof(char\*) \* strsSize);

        res[row][col] = (char \*)malloc(sizeof(char) \* strlen(strs[i]) + 1);

        strcpy(res[row][col++],strs[i]);//将当前行的字符串赋给三维数组第row层第col行

        for(j = i + 1; j < strsSize; j++) {

            if(strcmp(strscpy[i],strscpy[j]) == 0) {//第i行的字符串和后面每一行的字符串进行比较，如果相同

                book[j] = 1;//进行标记

                res[row][col] = (char \*)malloc(sizeof(char) \* strlen(strs[j]) + 1);

                strcpy(res[row][col++],strs[j]);//存到三维数组第row层第col行

            }

        }

        (\*returnColumnSizes)[row] = col;

        col = 0;//重新记录下一层的行数

        row++;//层数加1

    }

    \*returnSize = row;

    return res;

}

## [376. 摆动序列](https://leetcode-cn.com/problems/wiggle-subsequence/)

### 1 原题目

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为摆动序列。第一个差（如果存在的话）可能是正数或负数。少于两个元素的序列也是摆动序列。

例如， [1,7,4,9,2,5] 是一个摆动序列，因为差值 (6,-3,5,-7,3) 是正负交替出现的。相反, [1,4,7,2,5] 和 [1,7,4,5,5] 不是摆动序列，第一个序列是因为它的前两个差值都是正数，第二个序列是因为它的最后一个差值为零。

给定一个整数序列，返回作为摆动序列的最长子序列的长度。 通过从原始序列中删除一些（也可以不删除）元素来获得子序列，剩下的元素保持其原始顺序。

**示例 1:**

输入: [1,7,4,9,2,5]

输出: 6

解释: 整个序列均为摆动序列。

**示例 2:**

输入: [1,17,5,10,13,15,10,5,16,8]

输出: 7

解释: 这个序列包含几个长度为 7 摆动序列，其中一个可为[1,17,10,13,10,16,8]。

**示例 3:**

输入: [1,2,3,4,5,6,7,8,9]

输出: 2

**进阶:**

你能否用 O(n) 时间复杂度完成此题?

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/wiggle-subsequence>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C语言已过）

利用up记录前一时刻是上升情况的摆动序列的长度，down记录前一时刻是下降情况的摆动序列的长度，然后遍历数组求差值，将差值按照正负交替加入到这二者之一，最后在二者中选取一个最大值。

本人代码：

int wiggleMaxLength(int\* nums, int numsSize){

    if(numsSize <= 1)//少于两个元素的序列也是摆动序列

        return numsSize;

    int up = 1, down = 1;//up代表前一时刻上升，down代表前一时刻下降

    for(int i = 0;i < numsSize - 1; i++) {//遍历数组

        if(nums[i] - nums[i + 1] < 0)//如果此时差值是负数

            up = down + 1;//所以上升的加一（摆动序列要求正负交替出现）

        if(nums[i] - nums[i + 1] > 0)//如果此时差值是正数

            down = up + 1;//所以下降的加一（摆动序列要求正负交替出现）

    }

    return (up > down ? up : down);//返回最大值

}

## [面试题57 - II. 和为s的连续正数序列](https://leetcode-cn.com/problems/he-wei-sde-lian-xu-zheng-shu-xu-lie-lcof/)

### 1 原题目

输入一个正整数 target ，输出所有和为 target 的连续正整数序列（至少含有两个数）。

序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。

**示例 1：**

输入：target = 9

输出：[[2,3,4],[4,5]]

**示例 2：**

输入：target = 15

输出：[[1,2,3,4,5],[4,5,6],[7,8]]

**限制：**

1 <= target <= 10^5

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/he-wei-sde-lian-xu-zheng-shu-xu-lie-lcof>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

滑动窗口（解法一）

### 3 解法

#### 3.1 解法1（C++已过）

滑动窗口和双指针我始终搞不懂有什么区别，无语。。。。。。。

这道题就是用指针将连续的序列划分到一起（就像一个窗口，在左右指针之间的就是一个区间），我这道题将左右指针都包含了进去，然后将区间求和，求和之后通过比较然后将指针进行变换（说起来指针的变化确实像滑动的窗户，当然就是因为求连续的所以才像滑动窗口）。

本人代码：

class Solution {

public:

    vector<vector<int>> findContinuousSequence(int target) {

        vector<int>temp;

        vector<vector<int>>res;

        int left = 1;//左指针

        int right = 1;//右指针

        int sum = 1;//和（左右指针都包括）

        while(left <= target / 2) {//因为求的是连续正整数（所以左指针小于target的一半）

            if(sum < target) {//当和小于target

                right++;//右指针加一

                sum = sum + right;//求和

            }

            else if(sum > target) {//当和大于target

                sum = sum - left;//求和

                left++;//左指针加一

            }

            else {//当和等于target

                for(int i = left; i <= right; i++) {

                    temp.push\_back(i);//将结果存进temp

                }

                res.push\_back(temp);//将temp存进res

                temp.clear();//清空temp

                //计算下一种情况

                sum = sum - left;

                left++;

            }

        }

        return res;

    }

};

## [76. 最小覆盖子串](https://leetcode-cn.com/problems/minimum-window-substring/)

### 1 原题目

给你一个字符串 S、一个字符串 T，请在字符串 S 里面找出：包含 T 所有字符的最小子串。

**示例：**

输入: S = "ADOBECODEBANC", T = "ABC"

输出: "BANC"

**说明：**

如果 S 中不存这样的子串，则返回空字符串 ""。

如果 S 中存在这样的子串，我们保证它是唯一的答案。

### 2 基础知识

滑动窗口（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

滑动窗口解题，如果当前窗口不符合题意（T中所有字母在子串中出现）那么右指针向右，如果符合题意，左指针向右（缩小窗口就是为了找最小子串），重复这两步就可以。

本人代码：

|  |
| --- |
| class Solution {  public:      unordered\_map <char, int> tt,ss;        bool check() {//判断子串中是否包含T中所有字母          for (auto p:tt) {              if (ss[p.first] < p.second) {                  return false;              }          }          return true;      }      string minWindow(string s, string t) {          if(s.length() < t.length())              return "";          for(int i = 0; i < t.length(); i++)              tt[t[i]]++;          int left = 0,right = -1;          int ansl = -1,len = INT\_MAX;          //滑动窗口，如果窗口内子串不包含T中所有字母，那么右指针一直向右（扩大窗口）          //如果遍历到子串包含T中所有字母，那么左指针向右（缩小窗口），缩小的过程就是为了找最小子串，重复这两步          while(right < int(s.length())) {              right++;              ss[s[right]]++;              while(check() && left <= right) {                  if(len > right - left + 1) {                      ansl = left;                      len = right - left + 1;                  }                  ss[s[left]]--;                  left++;              }          }          return ansl == -1 ? "" : s.substr(ansl,len);      }  }; |

## [632. 最小区间](https://leetcode-cn.com/problems/smallest-range-covering-elements-from-k-lists/)

### 1 原题目

你有 k 个升序排列的整数数组。找到一个最小区间，使得 k 个列表中的每个列表至少有一个数包含在其中。

我们定义如果 b-a < d-c 或者在 b-a == d-c 时 a < c，则区间 [a,b] 比 [c,d] 小。

**示例 1:**

输入:[[4,10,15,24,26], [0,9,12,20], [5,18,22,30]]

输出: [20,24]

解释:

列表 1：[4, 10, 15, 24, 26]，24 在区间 [20,24] 中。

列表 2：[0, 9, 12, 20]，20 在区间 [20,24] 中。

列表 3：[5, 18, 22, 30]，22 在区间 [20,24] 中。

**注意:**

给定的列表可能包含重复元素，所以在这里升序表示 >= 。

1 <= k <= 3500

-105 <= 元素的值 <= 105

对于使用Java的用户，请注意传入类型已修改为List<List<Integer>>。重置代码模板后可以看到这项改动。

### 2 基础知识

哈希表+滑动窗口（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题目的解法有点类似于LeetCode 76题的解法，也不算类似，用的都是滑动窗口，可以说思想上是一样的（惭愧，看了题解才会做，哎）。此题之所以可以用滑动窗口的前提条件是要先建立一个哈希表，哈希表的键值对为：键:序列中的值，值:序列号。然后因为可以找到序列中的最大最小值，所以区间肯定在这范围内，所以我们可以进行滑动窗口，左窗口初始不动，右窗口一直像右直到所有的序列号都出现过一次（至少一次），此时说明在此窗口的范围内，所有序列至少有一个数字被包含在窗口内（找到了一组正确的区间，区间要找最短的才是答案）。此时右窗口不动，左窗口一直像右（这个过程就是根据哈希表减除序列号，右窗口像右则是加序列号）直到序列号有的没出现过。

本人代码：

|  |
| --- |
| class Solution {  public:      vector<int> smallestRange(vector<vector<int>>& nums) {          int n = nums.size();          //记录区间中的最大最小值          int maxx = INT\_MIN,minn = INT\_MAX;          unordered\_map<int,vector<int>> mp;          //找最大最小值，并且构建哈希表(键:序列中的值，值:序列号)          for(int i = 0; i < n; i++) {              for(int j = 0; j < nums[i].size(); j++) {                  maxx = max(maxx,nums[i][j]);                  minn = min(minn,nums[i][j]);                  mp[nums[i][j]].push\_back(i);              }          }          //当前滑动窗口的左端点和右端点          int nowleft = minn,nowright = minn - 1;          //结果的左右端点          int resleft = minn,resright = maxx;          //记录每个序列号出现的次数          vector<int> amo(n,0);          int qua = 0;//记录序列出现的个数          while(nowright < maxx) {              nowright++;//右端点右移              if(mp.count(nowright)) {//右端点在哈希表中存在                  for(int i = 0; i < mp[nowright].size(); i++) {//遍历哈希表                      amo[mp[nowright][i]]++;//序列号出现的次数加1                      if(amo[mp[nowright][i]] == 1) {//如果序列号出现过一次                          qua++;//序列个数加1                      }                  }                  while(qua == n) {//序列个数等于n(每个序列至少一个数字在当前窗口内)                      if(resright - resleft > nowright - nowleft) {//结果更新                          resright = nowright;                          resleft = nowleft;                      }                      if(mp.count(nowleft)) {//左端点在哈希表中存在                          for(int i = 0; i < mp[nowleft].size(); i++) {//遍历哈希表                              amo[mp[nowleft][i]]--;//序列号出现的次数减1                              if(amo[mp[nowleft][i]] == 0) {//如果序列号次数为0                                  qua--;//序列个数减1                              }                          }                      }                      nowleft++;//左端口右移                  }              }          }          return {resleft,resright};      }  }; |

## [面试题59 - II. 队列的最大值](https://leetcode-cn.com/problems/dui-lie-de-zui-da-zhi-lcof/)

### 1 原题目

请定义一个队列并实现函数 max\_value 得到队列里的最大值，要求函数max\_value、push\_back 和 pop\_front 的时间复杂度都是O(1)。

若队列为空，pop\_front 和 max\_value 需要返回 -1

**示例 1：**

输入:

["MaxQueue","push\_back","push\_back","max\_value","pop\_front","max\_value"]

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

输出: [null,null,null,2,1,2]

**示例 2：**

输入:

["MaxQueue","pop\_front","max\_value"]

[[],[],[]]

输出: [null,-1,-1]

**限制：**

1 <= push\_back,pop\_front,max\_value的总操作数 <= 10000

1 <= value <= 10^5

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/dui-lie-de-zui-da-zhi-lcof>

### 2 基础知识

#### 2.1数据结构

队列（解法一用到）

双向队列（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

本人参照的是官方题解的第二种方法：<https://leetcode-cn.com/problems/dui-lie-de-zui-da-zhi-lcof/solution/mian-shi-ti-59-ii-dui-lie-de-zui-da-zhi-by-leetcod/>

此题的难处在于求队列的最大值并要求时间复杂度为O(1)，此题的另外两个操作我们都可以通过队列直接实现，至于如何算出最大值这就需要用到双向队列这个数据结构。

队列是先进先出，所以给你如下的队列：1，1，1，1，2。1对于这个队列中的最大值2是没影响的，也就是说执行弹出操作时需要将1都删除最后才会删除2。

讲到这里我们如果在往队列中添加元素：

1. 添加一个3，3比2大。此时队列中3最大，在他前面的数字对他都构不成影响。
2. 添加一个1，1比2小，此时队列中仍然2最大，但是如果2删除了，队列中还会剩下一个1，此时这个1就是最大值了。

所以我们借助双向队列来求解元素，我上面说的这些，就是再向双向队列末尾添加新元素时，先进行比较，从双向队列末尾比较，如果小于当前元素的都舍去（第一种情况），直到碰到等于或大于的再添加进去，这样就保证了双向队列的头元素始终都是这个队列的最大值。

本人代码：

class MaxQueue {

queue<int>q;//队列

deque<int>m;//双向队列用来存储队列中的最大值

public:

MaxQueue() {

}

int max\_value() {

if(m.empty())

return -1;

return m.front();//从头部返回最大值

}

void push\_back(int value) {

q.push(value);//元素存进q中

while(!m.empty() && value > m.back())//当前元素和双向队列尾部比较

m.pop\_back();//小于当前元素的就删除

m.push\_back(value);//存进m的尾部

}

int pop\_front() {

if(q.empty())

return -1;

int res = q.front();

q.pop();

if(res == m.front())//如果队列中删除的元素等于双向队列中的头元素

m.pop\_front();//删除

return res;

}

};

/\*\*

\* Your MaxQueue object will be instantiated and called as such:

\* MaxQueue\* obj = new MaxQueue();

\* int param\_1 = obj->max\_value();

\* obj->push\_back(value);

\* int param\_3 = obj->pop\_front();

\*/

## [876. 链表的中间结点](https://leetcode-cn.com/problems/middle-of-the-linked-list/)

### 1 原题目

给定一个带有头结点 head 的非空单链表，返回链表的中间结点。

如果有两个中间结点，则返回第二个中间结点。

**示例 1：**

**输入：**[1,2,3,4,5]

**输出：**此列表中的结点 3 (序列化形式：[3,4,5])

返回的结点值为 3 。 (测评系统对该结点序列化表述是 [3,4,5])。

注意，我们返回了一个 ListNode 类型的对象 ans，这样：

ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, 以及 ans.next.next.next = NULL.

**示例 2：**

**输入：**[1,2,3,4,5,6]

**输出：**此列表中的结点 4 (序列化形式：[4,5,6])

由于该列表有两个中间结点，值分别为 3 和 4，我们返回第二个结点。

**提示：**

* 给定链表的结点数介于 1 和 100 之间。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

快慢指针（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

快慢指针，快指针每次走两步，慢指针每次走一步，要注意循环的终止条件。

本人代码：

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* middleNode(ListNode\* head) {

        ListNode \*fast = head;//快指针

        ListNode \*slow = head;//慢指针

        while(fast != NULL && fast -> next != NULL) {

            fast = fast -> next -> next;

            slow = slow -> next;

        }

        return slow;

    }

};

## [820. 单词的压缩编码](https://leetcode-cn.com/problems/short-encoding-of-words/)

### 1 原题目

给定一个单词列表，我们将这个列表编码成一个索引字符串 S 与一个索引列表 A。

例如，如果这个列表是 ["time", "me", "bell"]，我们就可以将其表示为 S = "time#bell#" 和 indexes = [0, 2, 5]。

对于每一个索引，我们可以通过从字符串 S 中索引的位置开始读取字符串，直到 "#" 结束，来恢复我们之前的单词列表。

那么成功对给定单词列表进行编码的最小字符串长度是多少呢？

**示例：**

输入: words = ["time", "me", "bell"]

输出: 10

说明: S = "time#bell#" ， indexes = [0, 2, 5] 。

**提示：**

1 <= words.length <= 2000

1 <= words[i].length <= 7

每个单词都是小写字母 。

### 2 基础知识

#### 2.1数据结构

前缀树（解法二用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

此题就是判断是否有字符串出现在其它字符串的末尾，出现了那么此字符串就可以被压缩。

所以我们可以先将字符串都反转存储并且按照字典序排序，然后我们就可以判断当前字符串是否是下一个字符串的开头部分（是就舍去，如果不是那么此字符串就为结果之一）

本人代码：

class Solution {

public:

    int minimumLengthEncoding(vector<string>& words) {

        vector<string>reverse\_words;

        for(string s : words) {//反转每个单词存到reverse\_words中

            reverse(s.begin(),s.end());

            reverse\_words.push\_back(s);

        }

        sort(reverse\_words.begin(), reverse\_words.end());//字典序排序

        int res = 0,len = words.size();

        for(int i = 0; i < len; i++) {

            if(i + 1 < len && reverse\_words[i + 1].find(reverse\_words[i]) == 0)//当前单词是下一个单词的前缀，丢弃

                continue;

            else

                res += reverse\_words[i].length() + 1;//单词长度加上一个 '#' 的长度

        }

        return res;

    }

};

#### 3.2 解法2（C++已过）

这道题是就是让我们剔除掉那些是其他单词后缀的单词。所以可以直接利用字典树来做就可。

注意单词插入的时候要倒着插入（因为此题是判断后缀）。

本人代码：

class node {

    public:

    bool torf = false;//代表此单词是否插入成功

    node \*next[26] = {nullptr};//每个节点有26个指针代表26个字母

};

class Trie {

public:

    node \*root;//树的根节点

    /\*\* Initialize your data structure here. \*/

    Trie() {

        root = new node();

    }

    /\*\* Inserts a word into the trie. \*/

    void insert(string word) {//将单词插入到前缀树中

        node \* temp = root;

        for(int i = word.length() - 1; i >= 0; i--) {

            if(temp -> next[word[i] - 'a'] == nullptr) {

                node \* t = new node();

                temp -> next[word[i] - 'a'] = t;

                temp = t;

            }

            else

                temp = temp -> next[word[i] - 'a'];

        }

        temp -> torf = true;

    }

};

//以上为前缀树的实现

class Solution {

public:

    int res = 0;

    void leaflen(node \*t,int len) {//通过递归来计算出所有从根节点到叶子节点的路径长度总和

        int flag = 0,i;

        for(i = 0; i < 26; i++) {

            if(t -> next[i] != nullptr) {

                flag = 1;

                leaflen(t -> next[i],len + 1);

            }

        }

        if(flag == 0) {

            res += len;

            return ;

        }

    }

    int minimumLengthEncoding(vector<string>& words) {

        Trie\* obj = new Trie();//前缀树

        for(string s : words) {//将单词插入进前缀树中

            obj -> insert(s);

        }

        leaflen(obj -> root,1);

        return res;

    }

};

## [208. 实现 Trie (前缀树)](https://leetcode-cn.com/problems/implement-trie-prefix-tree/)

### 1 原题目

实现一个 Trie (前缀树)，包含 insert, search, 和 startsWith 这三个操作。

**示例:**

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

**说明:**

你可以假设所有的输入都是由小写字母 a-z 构成的。

保证所有输入均为非空字符串。

### 2 基础知识

#### 2.1数据结构

前缀树（Trie）（解法一用到）

#### 2.2算法

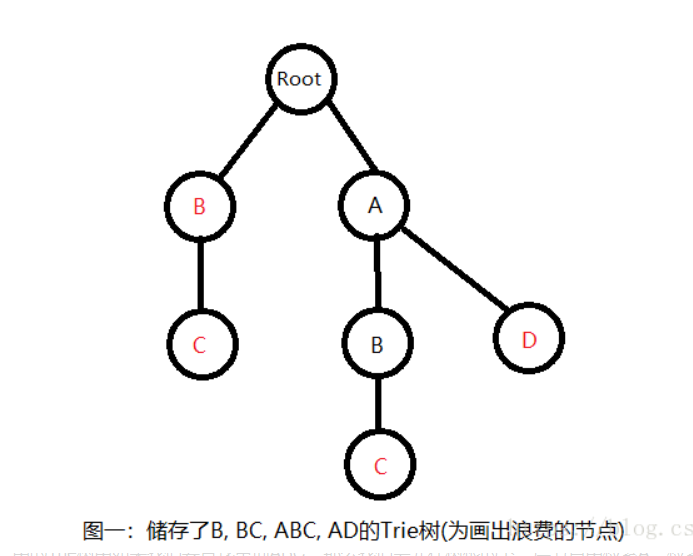
无。

### 3 解法

#### 3.1 解法1（C++已过）

自己看，不会去百度找。

前缀树实际上存储单词时是倒着存的，见下图。（所以是前缀树，哈哈哈，滑稽，手动狗头）



本人代码：

class node {

    public:

    bool torf = false;//代表此单词是否插入成功

    node \*next[26] = {nullptr};//每个节点有26个指针代表26个字母

};

class Trie {

    node \*root;//树的根节点

public:

    /\*\* Initialize your data structure here. \*/

    Trie() {

        root = new node();

    }

    /\*\* Inserts a word into the trie. \*/

    void insert(string word) {

        node \* temp = root;

        for(int i = 0; i < word.length(); i++) {

            if(temp -> next[word[i] - 'a'] == nullptr) {

                node \* t = new node();

                temp -> next[word[i] - 'a'] = t;

                temp = t;

            }

            else

                temp = temp -> next[word[i] - 'a'];

        }

        temp -> torf = true;

    }

    /\*\* Returns if the word is in the trie. \*/

    bool search(string word) {

        node \* temp = root;

        for(int i = 0; i < word.length(); i++) {

            if(temp -> next[word[i] - 'a'] != nullptr) {

                temp = temp -> next[word[i] - 'a'];

            }

            else

                return false;

        }

        return temp -> torf;

    }

    /\*\* Returns if there is any word in the trie that starts with the given prefix. \*/

    bool startsWith(string prefix) {

        node \* temp = root;

        for(int i = 0; i < prefix.length(); i++) {

            if(temp -> next[prefix[i] - 'a'] != nullptr) {

                temp = temp -> next[prefix[i] - 'a'];

            }

            else

                return false;

        }

        return true;

    }

};

/\*\*

 \* Your Trie object will be instantiated and called as such:

 \* Trie\* obj = new Trie();

 \* obj->insert(word);

 \* bool param\_2 = obj->search(word);

 \* bool param\_3 = obj->startsWith(prefix);

 \*/

## [336. 回文对](https://leetcode-cn.com/problems/palindrome-pairs/)

### 1 原题目

给定一组 互不相同 的单词， 找出所有不同 的索引对(i, j)，使得列表中的两个单词， words[i] + words[j] ，可拼接成回文串。

**示例 1：**

输入：["abcd","dcba","lls","s","sssll"]

输出：[[0,1],[1,0],[3,2],[2,4]]

解释：可拼接成的回文串为 ["dcbaabcd","abcddcba","slls","llssssll"]

**示例 2：**

输入：["bat","tab","cat"]

输出：[[0,1],[1,0]]

解释：可拼接成的回文串为 ["battab","tabbat"]

### 2 基础知识

前缀树（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此链接讲的很明白：<https://leetcode-cn.com/problems/palindrome-pairs/solution/hui-wen-dui-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class node {  public:      int torf = -1;//代表此单词的下标      node \*next[26] = {nullptr};//每个节点有26个指针代表26个字母  };  class Trie {  public:      node \*root;//树的根节点      /\*\* Initialize your data structure here. \*/      Trie() {          root = new node();      }      /\*\* Inserts a word into the trie. \*/      void insert(string word,int id) {//将单词插入到前缀树中          node \* temp = root;          for(int i = 0; i < word.length(); i++) {              if(temp -> next[word[i] - 'a'] == nullptr) {                  node \* t = new node();                  temp -> next[word[i] - 'a'] = t;                  temp = t;              }              else                  temp = temp -> next[word[i] - 'a'];          }          temp -> torf = id;      }      /\*\* Returns if there is any word in the trie that starts with the given prefix. \*/      //判断字典树中是否存在单词prefix并返回单词下标      int startsWith(string prefix) {          node \* temp = root;          for(int i = 0; i < prefix.length(); i++) {              if(temp -> next[prefix[i] - 'a'] != nullptr) {                  temp = temp -> next[prefix[i] - 'a'];              }              else                  return -1;          }          return temp -> torf;      }  };  class Solution {  public:      //判断s串的[left,right]部分是不是回文串       bool isPalindrome(string& s, int left, int right) {          int len = right - left + 1;          for (int i = 0; i < len / 2; i++) {              if (s[left + i] != s[right - i]) {                  return false;              }          }          return true;      }      vector<vector<int>> palindromePairs(vector<string>& words) {          //创建字典树          Trie\* trie = new Trie();          int n = words.size();          //将单词插入字典树中          for (int i = 0; i < n; i++)              trie -> insert(words[i],i);          vector<vector<int>> res;          //枚举每个单词的所有前缀和后缀并判断是否为回文串          //如果是回文串，我们就查询其剩余部分的翻转是否在给定的字符串序列中出现即可。          for (int i = 0; i < n; i++) {              int m = words[i].size();              for (int j = 0; j <= m; j++) {                  //前缀                  if (j && isPalindrome(words[i], 0, j - 1)) {                      //翻转剩余部分                      string temp = words[i].substr(j,m - j);                      reverse(temp.begin(),temp.end());                      //判断翻转后的剩余部分是否出现在单词集合words中                      int right\_id = trie -> startsWith(temp);                      if (right\_id != -1 && right\_id != i) {                          res.push\_back({right\_id, i});                      }                  }                  //后缀                  if (isPalindrome(words[i], j, m - 1)) {                      //翻转剩余部分                      string temp = words[i].substr(0,j);                      reverse(temp.begin(),temp.end());                      //判断翻转后的剩余部分是否出现在单词集合words中                      int left\_id = trie -> startsWith(temp);                      if (left\_id != -1 && left\_id != i) {                          res.push\_back({i, left\_id});                      }                  }              }          }          return res;      }  }; |

## [面试题62. 圆圈中最后剩下的数字](https://leetcode-cn.com/problems/yuan-quan-zhong-zui-hou-sheng-xia-de-shu-zi-lcof/)

### 1 原题目

0,1,,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字。求出这个圆圈里剩下的最后一个数字。

例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3。

**示例 1：**

输入: n = 5, m = 3

输出: 3

**示例 2：**

输入: n = 10, m = 17

输出: 2

**限制：**

1 <= n <= 10^5

1 <= m <= 10^6

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

数学+迭代（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

约瑟夫环问题。

推荐链接：<https://leetcode-cn.com/problems/yuan-quan-zhong-zui-hou-sheng-xia-de-shu-zi-lcof/solution/javajie-jue-yue-se-fu-huan-wen-ti-gao-su-ni-wei-sh/>

本人代码：

class Solution {

public:

    int lastRemaining(int n, int m) {

        int ans = 0;

        for(int i = 2; i <= n; i++)//最后一轮剩下2个人，所以从2开始反推

            ans = (ans + m) % i;

        return ans;

    }

};

## [面试题 16.03. 交点](https://leetcode-cn.com/problems/intersection-lcci/)

### 1 原题目

给定两条线段（表示为起点start = {X1, Y1}和终点end = {X2, Y2}），如果它们有交点，请计算其交点，没有交点则返回空值。

要求浮点型误差不超过10^-6。若有多个交点（线段重叠）则返回 X 值最小的点，X 坐标相同则返回 Y 值最小的点。

**示例 1：**

输入：

line1 = {0, 0}, {1, 0}

line2 = {1, 1}, {0, -1}

输出： {0.5, 0}

**示例 2：**

输入：

line1 = {0, 0}, {3, 3}

line2 = {1, 1}, {2, 2}

输出： {1, 1}

**示例 3：**

输入：

line1 = {0, 0}, {1, 1}

line2 = {1, 0}, {2, 1}

输出： {}，两条线段没有交点

**提示：**

坐标绝对值不会超过 2^7

输入的坐标均是有效的二维坐标

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/intersection-lcci>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

数学—线段相交（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

参考链接中的方法一：<https://leetcode-cn.com/problems/intersection-lcci/solution/jiao-dian-by-leetcode-solution/>（沉下心看，别毛躁）。

本人代码：

class Solution {

public:

    // 判断 (xk, yk) 是否在「线段」(x1, y1)~(x2, y2) 上

    // 这里的前提是 (xk, yk) 一定在「直线」(x1, y1)~(x2, y2) 上

    bool inside(int x1, int y1, int x2, int y2, int xk, int yk) {

        // 若与 x 轴平行，只需要判断 x 的部分

        // 若与 y 轴平行，只需要判断 y 的部分

        // 若为普通线段，则都要判断

        return (x1 == x2 || (min(x1, x2) <= xk && xk <= max(x1, x2))) && (y1 == y2 || (min(y1, y2) <= yk && yk <= max(y1, y2)));

    }

    void update(vector<double>& ans, double xk, double yk) {

        // 将一个交点与当前 ans 中的结果进行比较

        // 若更优则替换

        if (!ans.size() || xk < ans[0] || (xk == ans[0] && yk < ans[1])) {

            ans = {xk, yk};

        }

    }

    vector<double> intersection(vector<int>& start1, vector<int>& end1, vector<int>& start2, vector<int>& end2) {

        int x1 = start1[0], y1 = start1[1];

        int x2 = end1[0], y2 = end1[1];

        int x3 = start2[0], y3 = start2[1];

        int x4 = end2[0], y4 = end2[1];

        vector<double> ans;

        // 判断 (x1, y1)~(x2, y2) 和 (x3, y3)~(x4, y3) 是否平行

        if ((y4 - y3) \* (x2 - x1) == (y2 - y1) \* (x4 - x3)) {

            // 若平行，则判断 (x3, y3) 是否在「直线」(x1, y1)~(x2, y2) 上

            if ((y2 - y1) \* (x3 - x1) == (y3 - y1) \* (x2 - x1)) {

                // 判断 (x3, y3) 是否在「线段」(x1, y1)~(x2, y2) 上

                if (inside(x1, y1, x2, y2, x3, y3)) {

                    update(ans, (double)x3, (double)y3);

                }

                // 判断 (x4, y4) 是否在「线段」(x1, y1)~(x2, y2) 上

                if (inside(x1, y1, x2, y2, x4, y4)) {

                    update(ans, (double)x4, (double)y4);

                }

                // 判断 (x1, y1) 是否在「线段」(x3, y3)~(x4, y4) 上

                if (inside(x3, y3, x4, y4, x1, y1)) {

                    update(ans, (double)x1, (double)y1);

                }

                // 判断 (x2, y2) 是否在「线段」(x3, y3)~(x4, y4) 上

                if (inside(x3, y3, x4, y4, x2, y2)) {

                    update(ans, (double)x2, (double)y2);

                }

            }

            // 在平行时，其余的所有情况都不会有交点

        }

        else {

            // 联立方程得到 t1 和 t2 的值

            double t1 = (double)(x3 \* (y4 - y3) + y1 \* (x4 - x3) - y3 \* (x4 - x3) - x1 \* (y4 - y3)) / ((x2 - x1) \* (y4 - y3) - (x4 - x3) \* (y2 - y1));

            double t2 = (double)(x1 \* (y2 - y1) + y3 \* (x2 - x1) - y1 \* (x2 - x1) - x3 \* (y2 - y1)) / ((x4 - x3) \* (y2 - y1) - (x2 - x1) \* (y4 - y3));

            // 判断 t1 和 t2 是否均在 [0, 1] 之间

            if (t1 >= 0.0 && t1 <= 1.0 && t2 >= 0.0 && t2 <= 1.0) {

                ans = {x1 + t1 \* (x2 - x1), y1 + t1 \* (y2 - y1)};

            }

        }

        return ans;

    }

};

## [460. LFU缓存](https://leetcode-cn.com/problems/lfu-cache/)

### 1 原题目

请你为 最不经常使用（LFU）缓存算法设计并实现数据结构。它应该支持以下操作：get 和 put。

get(key) - 如果键存在于缓存中，则获取键的值（总是正数），否则返回 -1。

put(key, value) - 如果键不存在，请设置或插入值。当缓存达到其容量时，则应该在插入新项之前，使最不经常使用的项无效。在此问题中，当存在平局（即两个或更多个键具有相同使用频率）时，应该去除 最近 最少使用的键。

「项的使用次数」就是自插入该项以来对其调用 get 和 put 函数的次数之和。使用次数会在对应项被移除后置为 0 。

**进阶：**

你是否可以在 O(1) 时间复杂度内执行两项操作？

**示例：**

LFUCache cache = new LFUCache( 2 /\* capacity (缓存容量) \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // 返回 1

cache.put(3, 3); // 去除 key 2

cache.get(2); // 返回 -1 (未找到key 2)

cache.get(3); // 返回 3

cache.put(4, 4); // 去除 key 1

cache.get(1); // 返回 -1 (未找到 key 1)

cache.get(3); // 返回 3

cache.get(4); // 返回 4

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

这道题就是利用两个哈希表，一个是用key值作为键，一个是用使用频率作键。两个哈希表像横纵坐标一样可以锁定如果满了之后删除哪个元素，两个哈希表的值都是链表类型，可以通过头尾的操作达到当使用频次相同时按时间删除。

参考链接：<https://leetcode-cn.com/problems/lfu-cache/solution/lfuhuan-cun-by-leetcode-solution/>

这道题用到了unordered\_map，构造函数初始化结构体，迭代器，链表，可以好好看看，会有很大的收获。

本人代码：

//缓存的节点信息

struct Node {

    int key, val, freq;//键，值，使用频率

    Node(int \_key,int \_val,int \_freq): key(\_key), val(\_val), freq(\_freq){}//构造函数用来初始化结构体中的值

};

class LFUCache {

    int minfreq, capacity;//最小使用频率，缓存容量

    //这两个unordered\_map类似于二维数组的横纵坐标（官方解的图可以看出）,所以一个改变另一个必将改变

    unordered\_map<int, list<Node>::iterator> key\_table;//key值为键，iterator为迭代器（所以值的类型是迭代器）

    unordered\_map<int, list<Node>> freq\_table;//频率为键

public:

    LFUCache(int \_capacity) {

        minfreq = 0;

        capacity = \_capacity;

        key\_table.clear();//清除内容

        freq\_table.clear();//清除内容

    }

    int get(int key) {

        if (capacity == 0) return -1;

        auto it = key\_table.find(key);//find函数返回值是指向键等于 key 的元素的迭代器。若找不到这种元素，则返回尾后迭代器

        if (it == key\_table.end()) return -1;

        list<Node>::iterator node = it -> second;

        int val = node -> val, freq = node -> freq;

        freq\_table[freq].erase(node);//从频率为freq中移除键为key的元素（get操作增加一次访问次数）

        // 如果当前链表为空，我们需要在哈希表中删除，且更新minFreq

        if (freq\_table[freq].size() == 0) {

            freq\_table.erase(freq);

            if (minfreq == freq) minfreq += 1;

        }

        // 插入到 freq + 1 中

        freq\_table[freq + 1].push\_front(Node(key, val, freq + 1));//从链表的头部加入进去

        key\_table[key] = freq\_table[freq + 1].begin();//更新

        return val;

    }

    void put(int key, int value) {

        if (capacity == 0) return;

        auto it = key\_table.find(key);

        if (it == key\_table.end()) {

            // 缓存已满，需要进行删除操作

            if (key\_table.size() == capacity) {

                // 通过 minFreq 拿到 freq\_table[minFreq] 链表的末尾节点

                auto it2 = freq\_table[minfreq].back();

                key\_table.erase(it2.key);//从key\_table中删除最近使用最少次数的

                freq\_table[minfreq].pop\_back();//从freq\_table的链表中删除最后一项

                if (freq\_table[minfreq].size() == 0) {

                    freq\_table.erase(minfreq);

                }

            }

            freq\_table[1].push\_front(Node(key, value, 1));//新加入的使用频率为1，所以加到1的链表的头部

            key\_table[key] = freq\_table[1].begin();

            minfreq = 1;

        } else {

            // 与 get 操作基本一致，除了需要更新缓存的值

            list<Node>::iterator node = it -> second;

            int freq = node -> freq;

            freq\_table[freq].erase(node);

            if (freq\_table[freq].size() == 0) {

                freq\_table.erase(freq);

                if (minfreq == freq) minfreq += 1;

            }

            freq\_table[freq + 1].push\_front(Node(key, value, freq + 1));

            key\_table[key] = freq\_table[freq + 1].begin();

        }

    }

};

/\*\*

 \* Your LFUCache object will be instantiated and called as such:

 \* LFUCache\* obj = new LFUCache(capacity);

 \* int param\_1 = obj->get(key);

 \* obj->put(key,value);

 \*/

## [146. LRU缓存机制](https://leetcode-cn.com/problems/lru-cache/)

### 1 原题目

运用你所掌握的数据结构，设计和实现一个  LRU (最近最少使用) 缓存机制。它应该支持以下操作： 获取数据 get 和 写入数据 put 。

获取数据 get(key) - 如果密钥 (key) 存在于缓存中，则获取密钥的值（总是正数），否则返回 -1。

写入数据 put(key, value) - 如果密钥已经存在，则变更其数据值；如果密钥不存在，则插入该组「密钥/数据值」。当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。

**进阶:**

你是否可以在 O(1) 时间复杂度内完成这两种操作？

**示例:**

LRUCache cache = new LRUCache( 2 /\* 缓存容量 \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // 返回 1

cache.put(3, 3); // 该操作会使得密钥 2 作废

cache.get(2); // 返回 -1 (未找到)

cache.put(4, 4); // 该操作会使得密钥 1 作废

cache.get(1); // 返回 -1 (未找到)

cache.get(3); // 返回 3

cache.get(4); // 返回 4

### 2 基础知识

哈希表（解法一用到）

双向链表（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用双向链表，最近一次使用的从链表中找到并将它移动到链表头部，这样就保证了链表尾部是最近没使用的。然后利用哈希表记录键值所对应的在链表中的位置（降低时间复杂度），然后你就自己看注释吧，哈哈哈哈。

本人代码：

|  |
| --- |
| struct DLinkedNode {      int key, val;      DLinkedNode\* prev;      DLinkedNode\* next;      DLinkedNode(): key(0), val(0), prev(nullptr), next(nullptr) {}      DLinkedNode(int \_key, int \_val): key(\_key), val(\_val), prev(nullptr), next(nullptr) {}  };  class LRUCache {  public:      unordered\_map<int, DLinkedNode\*> cache;      DLinkedNode\* head;      DLinkedNode\* tail;      int size;      LRUCache(int capacity) {          //使用伪头部和伪尾部节点          head = new DLinkedNode();          tail = new DLinkedNode();          head->next = tail;          tail->prev = head;          size = capacity;      }        int get(int key) {          if(cache.find(key) == cache.end())              return -1;          //如果key存在，先通过哈希表定位，再移到头部          DLinkedNode\* node = cache[key];          removeNode(node);          addToHead(node);          return node -> val;      }        void put(int key, int value) {          if(cache.find(key) == cache.end()) {              //如果key不存在，创建一个新的节点              DLinkedNode\* node = new DLinkedNode(key, value);              //添加进哈希表              cache[key] = node;              //添加至双向链表的头部              addToHead(node);              size--;              if(size < 0) {                  //如果超出容量，删除双向链表的尾部节点                  DLinkedNode\* removed = tail -> prev;                  removeNode(removed);                  //删除哈希表中对应的项                  cache.erase(removed -> key);                  size++;              }          }          else {              //如果key存在，先通过哈希表定位，再修改value，并移到头部              DLinkedNode\* node = cache[key];              node -> val = value;              removeNode(node);              addToHead(node);          }      }      void removeNode(DLinkedNode\* node) {//从双向链表中移除一个节点          node -> prev -> next = node -> next;          node -> next -> prev = node -> prev;      }      void addToHead(DLinkedNode\* node) {//将一个节点加到头部          node -> prev = head;          node -> next = head -> next;          head -> next -> prev = node;          head -> next = node;      }  };  /\*\*   \* Your LRUCache object will be instantiated and called as such:   \* LRUCache\* obj = new LRUCache(capacity);   \* int param\_1 = obj->get(key);   \* obj->put(key,value);   \*/ |

## [542. 01矩阵](https://leetcode-cn.com/problems/01-matrix/)

### 1 原题目

给定一个由 0 和 1 组成的矩阵，找出每个元素到最近的 0 的距离。

两个相邻元素间的距离为 1 。

**示例 1:**

输入:

0 0 0

0 1 0

0 0 0

输出:

0 0 0

0 1 0

0 0 0

**示例 2:**

输入:

0 0 0

0 1 0

1 1 1

输出:

0 0 0

0 1 0

1 2 1

**注意:**

给定矩阵的元素个数不超过 10000。

给定矩阵中至少有一个元素是 0。

矩阵中的元素只在四个方向上相邻: 上、下、左、右。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

因为只能按照上下左右四个方向，所以我们可以利用动态规划来解题。

i,j位置的最优解一定是这四个方向的最优解加1，因为0元素到0的距离就是0，所以部分res数组中元素赋值为0（已经是最优解），让我们可以进行动态规划用已知的最优解求出最优解。

本人代码：

class Solution {

public:

    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {

        int m = matrix.size(), n = matrix[0].size();

        vector<vector<int>> res(m, vector<int>(n,INT\_MAX / 2));//开辟m行n列初始值为整形最大值的一半

        int i,j;

        for(i = 0; i < m; i++)

            for(j = 0; j < n; j++)

                if(matrix[i][j] == 0)//元素0到最近0的距离就是0

                    res[i][j] = 0;

        //只有水平向左移动和竖直向上移动

        for(i = 0; i < m; i++)

            for(j = 0; j < n; j++) {

                if(i - 1 >= 0)//从上面到i,j位置

                    res[i][j] = min(res[i][j],res[i - 1][j] + 1);

                if(j - 1 >= 0)//从左面到i,j位置

                    res[i][j] = min(res[i][j],res[i][j - 1] + 1);

            }

        //只有水平向右移动和竖直向下移动

        for(i = m - 1; i >= 0; i--)

            for(j = n - 1; j >= 0; j--) {

                if(i + 1 < m)//从下面到i,j位置

                    res[i][j] = min(res[i][j],res[i + 1][j] + 1);

                if(j + 1 < n)//从右面到i,j位置

                    res[i][j] = min(res[i][j],res[i][j + 1] + 1);

            }

        return res;

    }

};

## [55. 跳跃游戏](https://leetcode-cn.com/problems/jump-game/)

### 1 原题目

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

**示例 1:**

输入: [2,3,1,1,4]

输出: true

解释: 我们可以先跳 1 步，从位置 0 到达 位置 1, 然后再从位置 1 跳 3 步到达最后一个位置。

**示例 2:**

输入: [3,2,1,0,4]

输出: false

解释: 无论怎样，你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0 ， 所以你永远不可能到达最后一个位置。

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/jump-game>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

贪心算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用贪心的思想，在当前可以走到的步数中寻找新的最远距离并不断更新。

本人代码：

|  |
| --- |
| class Solution {  public:      bool canJump(vector<int>& nums) {          int n = nums.size();          int maxx = 0;//记录当前可以到达的最大下标值          for (int i = 0; i < n; i++) {              if (i <= maxx) {//下标小于当前最大下标值意味着可以到达                  maxx = max(maxx, i + nums[i]);//更新最大到达下标值                  if (maxx >= n - 1) {//超出数组长度                      return true;                  }              }          }          return false;      }  }; |

## [466. 统计重复个数](https://leetcode-cn.com/problems/count-the-repetitions/)

### 1 原题目

由 n 个连接的字符串 s 组成字符串 S，记作 S = [s,n]。例如，["abc",3]=“abcabcabc”。

如果我们可以从 s2 中删除某些字符使其变为 s1，则称字符串 s1 可以从字符串 s2 获得。例如，根据定义，"abc" 可以从 “abdbec” 获得，但不能从 “acbbe” 获得。

现在给你两个非空字符串 s1 和 s2（每个最多 100 个字符长）和两个整数 0 ≤ n1 ≤ 106 和 1 ≤ n2 ≤ 106。现在考虑字符串 S1 和 S2，其中 S1=[s1,n1] 、S2=[s2,n2] 。

请你找出一个可以满足使[S2,M] 从 S1 获得的最大整数 M 。

**示例：**

输入：

s1 ="acb",n1 = 4

s2 ="ab",n2 = 2

返回：

2

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/count-the-repetitions>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

这道题的思路就是寻找循环节。

参考链接：<https://leetcode-cn.com/problems/count-the-repetitions/solution/tong-ji-zhong-fu-ge-shu-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      int getMaxRepetitions(string s1, int n1, string s2, int n2) {          if (n1 == 0)              return 0;          int s1cnt = 0, index = 0, s2cnt = 0;          //注意s2要从第index个字符开始匹配          unordered\_map<int, pair<int, int>> recall;          pair<int, int> pre\_loop, in\_loop;          while (true) {              //我们多遍历一个s1，看看能不能找到循环节(鸽笼原理)              s1cnt++;              for (int i = 0; i < s1.length(); i++) {                  if (s1[i] == s2[index]) {                      index++;                      if (index == s2.length()) {                          s2cnt++;                          index = 0;                      }                  }              }              //还没有找到循环节，所有的s1就用完了              if (s1cnt == n1)                  return s2cnt / n2;              //出现了之前的index，表示找到了循环节              if (recall.count(index)) {                  auto [s1cnt\_prime, s2cnt\_prime] = recall[index];                  // 前 s1cnt' 个 s1 包含了 s2cnt' 个 s2                  pre\_loop = {s1cnt\_prime, s2cnt\_prime};                  // 以后的每 (s1cnt - s1cnt') 个 s1 包含了 (s2cnt - s2cnt') 个 s2                  in\_loop = {s1cnt - s1cnt\_prime, s2cnt - s2cnt\_prime};                  break;              }              else {                  recall[index] = {s1cnt, s2cnt};              }          }          // ans 存储的是 S1 包含的 s2 的数量，考虑的之前的 pre\_loop 和 in\_loop          int ans = pre\_loop.second + (n1 - pre\_loop.first) / in\_loop.first \* in\_loop.second;          // S1 的末尾还剩下一些 s1，我们暴力进行匹配          int rest = (n1 - pre\_loop.first) % in\_loop.first;          for (int i = 0; i < rest; ++i) {              for (int i = 0; i < s1.length(); i++) {                  if (s1[i] == s2[index]) {                      index++;                      if (index == s2.length()) {                          ans++;                          index = 0;                      }                  }              }          }          //S1包含ans个s2，那么就包含ans / n2个S2          return ans / n2;      }  }; |

## [1248. 统计「优美子数组」](https://leetcode-cn.com/problems/count-number-of-nice-subarrays/)

### 1 原题目

给你一个整数数组 nums 和一个整数 k。

如果某个 连续 子数组中恰好有 k 个奇数数字，我们就认为这个子数组是「优美子数组」。

请返回这个数组中「优美子数组」的数目。

**示例 1：**

输入：nums = [1,1,2,1,1], k = 3

输出：2

解释：包含 3 个奇数的子数组是 [1,1,2,1] 和 [1,2,1,1] 。

**示例 2：**

输入：nums = [2,4,6], k = 1

输出：0

解释：数列中不包含任何奇数，所以不存在优美子数组。

**示例 3：**

输入：nums = [2,2,2,1,2,2,1,2,2,2], k = 2

输出：16

**提示：**

1 <= nums.length <= 50000

1 <= nums[i] <= 10^5

1 <= k <= nums.length

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/count-number-of-nice-subarrays>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

数学（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

我们可以将数组中出现奇数的下标存到数组odd中，这样就知道odd中区间[i,i+k-1]含k个奇数。我们只需求出左边界左边偶数个数和右边界右边偶数的个数就可以求出包含k个奇数有多少个区间。

~~~~start-----end~~~~ [start,end]包含k个奇数

求出 start 左边偶数个数，得到左侧 left

求出 end 右侧偶数个数，得到右侧 right

一个 K 长度就有组合：

1.~~~~start-----end：有 left个

2.start-----end：有 1 个

3.~~~~start-----end~~~~：有 left \* right 个

4.start-----end~~~~：有 right 个

所以含k个奇数的区间就有 left + right + left \* right + 1 个。

本人代码：

|  |
| --- |
| class Solution {  public:      int numberOfSubarrays(vector<int>& nums, int k) {          int odd[nums.size() + 2],ans = 0,cnt = 1,i;          for (i = 0; i < nums.size(); i++) {//将奇数出现的下标存储在odd中              if (nums[i] % 2 != 0)                  odd[cnt++] = i;          }          odd[0] = -1;          odd[cnt] = nums.size();          int left,right;//记录区间左边偶数数量和右边偶数数量          for (i = 1; i + k <= cnt; i++) {              left = odd[i] - odd[i - 1] - 1;              right = odd[i + k] - odd[i + k - 1] - 1;              ans += 1 + left + right + left \* right;          }          return ans;      }  }; |

## [面试题51. 数组中的逆序对](https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof/)

### 1 原题目

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。

**示例 1:**

输入: [7,5,6,4]

输出: 5

**限制：**

0 <= 数组长度 <= 50000

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof>

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

归并排序（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

利用归并排序不断将数组划分成小数组在重新合并的这个过程，在合并过程中可以通过下标的计算算出逆序对。（如下图所示）



上图链接：<https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof/solution/shu-zu-zhong-de-ni-xu-dui-by-leetcode-solution/>

本人代码：

|  |
| --- |
| int ans;  //归并排序  void MergeSort(int \*R,int \*T,int low,int high)  {      //R[low..high]归并排序后放入T[low..high]      if(low == high)          T[low] = R[low];      else      {          int mid = (low + high) / 2;//将当前数组一分为二，求出分裂点mid          MergeSort(R,T,low,mid);//对子数组R[low..mid]递归归并排序，结果放入T[low..mid]          MergeSort(R,T,mid + 1,high);//对子数组R[mid+1..high]递归归并排序，结果放入T[mid+1..high]          //将有序数组R[low..mid]和R[mid+1..high]归并为有序数组T[low..high]          int i = low,j = mid + 1,k = low;          while(i <= mid && j <= high)//将R中的记录由小到大地并入T中          {              if(R[i] <= R[j])                  T[k++] = R[i++];              else {                  T[k++] = R[j++];                  ans += (mid - i + 1);//新添加的一行代码（计算逆序对的个数）              }          }          while(i <= mid)//将剩余的R[i..mid]复制到T中              T[k++] = R[i++];          while(j <= high)//将剩余的R[j..high]复制到T中              T[k++] = R[j++];          //将有序数组T[low,k-1]赋值给R[low,k-1]          for(j = low; j < k; j++)              R[j] = T[j];      }  }  int reversePairs(int\* nums, int numsSize){      if(numsSize == 0)          return 0;      ans = 0;      int \*T = (int \*)malloc(sizeof(int) \* numsSize);      MergeSort(nums,T,0,numsSize - 1);      return ans;  } |

## [983. 最低票价](https://leetcode-cn.com/problems/minimum-cost-for-tickets/)

### 1 原题目

在一个火车旅行很受欢迎的国度，你提前一年计划了一些火车旅行。在接下来的一年里，你要旅行的日子将以一个名为 days 的数组给出。每一项是一个从 1 到 365 的整数。

火车票有三种不同的销售方式：

一张为期一天的通行证售价为 costs[0] 美元；

一张为期七天的通行证售价为 costs[1] 美元；

一张为期三十天的通行证售价为 costs[2] 美元。

通行证允许数天无限制的旅行。 例如，如果我们在第 2 天获得一张为期 7 天的通行证，那么我们可以连着旅行 7 天：第 2 天、第 3 天、第 4 天、第 5 天、第 6 天、第 7 天和第 8 天。

返回你想要完成在给定的列表 days 中列出的每一天的旅行所需要的最低消费。

**示例 1：**

输入：days = [1,4,6,7,8,20], costs = [2,7,15]

输出：11

解释：

例如，这里有一种购买通行证的方法，可以让你完成你的旅行计划：

在第 1 天，你花了 costs[0] = $2 买了一张为期 1 天的通行证，它将在第 1 天生效。

在第 3 天，你花了 costs[1] = $7 买了一张为期 7 天的通行证，它将在第 3, 4, ..., 9 天生效。

在第 20 天，你花了 costs[0] = $2 买了一张为期 1 天的通行证，它将在第 20 天生效。

你总共花了 $11，并完成了你计划的每一天旅行。

**示例 2：**

输入：days = [1,2,3,4,5,6,7,8,9,10,30,31], costs = [2,7,15]

输出：17

解释：

例如，这里有一种购买通行证的方法，可以让你完成你的旅行计划：

在第 1 天，你花了 costs[2] = $15 买了一张为期 30 天的通行证，它将在第 1, 2, ..., 30 天生效。

在第 31 天，你花了 costs[0] = $2 买了一张为期 1 天的通行证，它将在第 31 天生效。

你总共花了 $17，并完成了你计划的每一天旅行。

**提示：**

1 <= days.length <= 365

1 <= days[i] <= 365

days 按顺序严格递增

costs.length == 3

1 <= costs[i] <= 1000

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

记忆化搜索（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

记忆化搜索自顶向下进行计算。

对于一年中的任何一天：

大于365，花费为0；

在days数组中，就要从三种买票方式中选择最便宜的一种；

不在days数组中，不需要买票，所以花费不变（就是等于后一天的花费）

参考链接：<https://leetcode-cn.com/problems/minimum-cost-for-tickets/solution/zui-di-piao-jie-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {      unordered\_set<int> dayset;      vector<int> costs;      int money[366] = {0};//代表第i天到最后一天花费的钱数  public:      int mincostTickets(vector<int>& days, vector<int>& costs) {          this -> costs = costs;          for(int i = 0; i < days.size(); i++)              dayset.insert(days[i]);          memset(money,-1,sizeof(money));          return dp(0);      }      int dp(int i) {          if(i > 365)//超过365花费肯定是0元              return 0;          if(money[i] != -1)              return money[i];          if(dayset.count(i))//第i天在days数组中，选取三种花费最小的              money[i] = min(min(dp(i + 1) + costs[0],dp(i + 7) + costs[1]),dp(i + 30) + costs[2]);          else//第i天不在days数组中，就不用买票，所以等于后一天的钱数              money[i] = dp(i + 1);          return money[i];      }  }; |

## [560. 和为K的子数组](https://leetcode-cn.com/problems/subarray-sum-equals-k/)

### 1 原题目

给定一个整数数组和一个整数 k，你需要找到该数组中和为 k 的连续的子数组的个数。

**示例 1 :**

输入:nums = [1,1,1], k = 2

输出: 2 , [1,1] 与 [1,1] 为两种不同的情况。

**说明 :**

数组的长度为 [1, 20,000]。

数组中元素的范围是 [-1000, 1000] ，且整数 k 的范围是 [-1e7, 1e7]。

### 2 基础知识

#### 2.1数据结构

前缀和+哈希表（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

利用了前缀和和哈希表来解题。

假设pre[i]为[0..i]里所有数的和，则pre[i]可以由pre[i−1]递推而来，pre[i]=pre[i−1]+nums[i]

那么[j..i]这个子数组和为k这个条件我们可以转化为pre[i]−pre[j−1]==k，再进行转化为pre[j−1]==pre[i]−k

 所以我们就可以在当前元素之前的所有前缀和中找是否有等于pre[i]−k的，有几个等的就加几个到结果中。

而前缀和直接可以利用常量来运算就可以，并且我们是从前到后进行遍历，可以保证区间不会有重复的，而且借用的只是之前前缀和出现的次数（在哈希表中存储）和当前的前缀和。

这题就是公示的转换想得到的话就会优化，想不到暴力的话就会超时。

本人代码：

|  |
| --- |
| class Solution {  public:      int subarraySum(vector<int>& nums, int k) {          unordered\_map<int, int> mp;//记录每个总和出现的次数          mp[0] = 1;          int res = 0,sum = 0;//sum记录num[0]到nums[i]的和          //假设pre[i]为[0..i]里所有数的和，则pre[i]可以由pre[i−1]递推而来，pre[i]=pre[i−1]+nums[i]          //那么[j..i]这个子数组和为k这个条件我们可以转化为pre[i]−pre[j−1]==k，再进行转化为pre[j−1]==pre[i]−k          //所以我们就可以在当前元素之前的所有前缀和中找是否有等于pre[i]−k的，有几个等的就加几个到结果中          for(int i = 0; i < nums.size(); i++) {              sum += nums[i];              if(mp.find(sum - k) != mp.end())                  res += mp[sum - k];              mp[sum]++;          }          return res;      }  }; |

## [974. 和可被 K 整除的子数组](https://leetcode-cn.com/problems/subarray-sums-divisible-by-k/)

### 1 原题目

给定一个整数数组 A，返回其中元素之和可被 K 整除的（连续、非空）子数组的数目。

**示例：**

输入：A = [4,5,0,-2,-3,1], K = 5

输出：7

解释：

有 7 个子数组满足其元素之和可被 K = 5 整除：

[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]

**提示：**

1 <= A.length <= 30000

-10000 <= A[i] <= 10000

2 <= K <= 10000

### 2 基础知识

哈希表（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题和560题有异曲同工之妙，都是遍历一次，利用哈希表记录下值，来计算最后结果，此题推导出可以利用哈希表是因为同余定理。

pre[i]为A[0]到A[i]的和，则sum(i,j) = pre[j] - pre[i - 1];

若(pre[j] - pre[i - 1]) % k == 0,那么由同余定理可知

pre[j] % k == pre[i - 1] % k

上面三行就是推导过程，推导完成后发现用哈希表记录余数就可以。但还有一点需要注意，此题我的代码余数全部转为正数了，这是为了可以一步求解结果。

就本题而言，比如说K=5,当前的的sum=4,那么4可以和之前出现的-1组合，也是5的倍数。-1取余还是-1，但是（-1+5)取余是4，4和4也可以组合，不影响计算结果。当然也可以不纠正成正数，保留(-K,K)的范围，计算的时候算两个，4需要统计4、-1的结果，稍微会麻烦

一点。

本人代码：

|  |
| --- |
| class Solution {  public:      int subarraysDivByK(vector<int>& A, int K) {          unordered\_map<int, int> mp;//记录每个总和对K求余后余数出现的次数          mp[0] = 1;          int sum = 0,res = 0,i;//sum记录A[0]到A[i]的和对K求余后的余数          for(i = 0; i < A.size(); i++) {              sum = ((sum + A[i]) % K + K) % K;//余数要保证全为正数，方便下面计算              //pre[i]为A[0]到A[i]的和，则sum(i,j) = pre[j] - pre[i - 1];              //若(pre[j] - pre[i - 1]) % k == 0,那么由同余定理可知              //pre[j] % k == pre[i - 1] % k              if(mp.find(sum) != mp.end())                  res += mp[sum];              mp[sum]++;          }          return res;      }  }; |

## [1371. 每个元音包含偶数次的最长子字符串](https://leetcode-cn.com/problems/find-the-longest-substring-containing-vowels-in-even-counts/)

### 1 原题目

给你一个字符串 s ，请你返回满足以下条件的最长子字符串的长度：每个元音字母，即 'a'，'e'，'i'，'o'，'u' ，在子字符串中都恰好出现了偶数次

**示例 1：**

输入：s = "eleetminicoworoep"

输出：13

解释：最长子字符串是 "leetminicowor" ，它包含 e，i，o 各 2 个，以及 0 个 a，u 。

**示例 2：**

输入：s = "leetcodeisgreat"

输出：5

解释：最长子字符串是 "leetc" ，其中包含 2 个 e 。

**示例 3：**

输入：s = "bcbcbc"

输出：6

解释：这个示例中，字符串 "bcbcbc" 本身就是最长的，因为所有的元音 a，e，i，o，u 都出现了 0 次。

**提示：**

1 <= s.length <= 5 x 10^5

s 只包含小写英文字母。

### 2 基础知识

前缀和+状态压缩（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

这种求子串的问题想到要用前缀和，但是我没想到状态压缩。

本题就是求一个最长子串（里面的每个元音字母都出现偶数次）。前缀和就是统计从0到下标i时每个元音字母出现的次数，但是因为暴力双重for循环会超时，所以要另外想办法。

不得不服官方题解，因为题目中的要求是偶数次，所以我们只需要记录每个字母的出现的次数是奇数次还是偶数次就可以了（奇数减奇数等于偶数，偶数减偶数等于偶数），这时候就可以用0代表出现次数为偶数，1代表出现次数为奇数。这时候发现就可以利用状态压缩了，因为要想子串满足题意，必须后面的状态和前面某一个状态一样。因为只需0或1表示出现次数，又因为元音字母5个，所以是二进制表示的(00000)到(11111)，转成十进制就是0到31，所以可以用数组存储元音字母出现的状态时字符串的长度（下标0到当前位置），然后一遍for循环就可以解出来结果。这中间运算需要位运算，毕竟压缩的状态是从二进制压缩到十进制，所以位运算更改状态容易些。

参考链接：<https://leetcode-cn.com/problems/find-the-longest-substring-containing-vowels-in-even-counts/solution/mei-ge-yuan-yin-bao-han-ou-shu-ci-de-zui-chang-z-2/>

本人代码：

|  |
| --- |
| class Solution {  public:      int findTheLongestSubstring(string s) {          int len = s.length(),res = 0,status = 0;          vector<int>pos(1<<5,-1);//存储的是字符串的长度          pos[0] = 0;          for(int i = 0; i < len; i++) {              if(s[i] ==  'a')                  status ^= 1<<0;              if(s[i] ==  'e')                  status ^= 1<<1;              if(s[i] ==  'i')                  status ^= 1<<2;              if(s[i] ==  'o')                  status ^= 1<<3;              if(s[i] ==  'u')                  status ^= 1<<4;              if(pos[status] != -1)                  res = max(res,i + 1 - pos[status]);              else                  pos[status] = i + 1;          }          return res;      }  }; |

## [5. 最长回文子串](https://leetcode-cn.com/problems/longest-palindromic-substring/)

### 1 原题目

给定一个字符串 s，找到 s 中最长的回文子串。你可以假设 s 的最大长度为 1000。

**示例 1：**

输入: "babad"

输出: "bab"

注意: "aba" 也是一个有效答案。

**示例 2：**

输入: "cbbd"

输出: "bb"

### 2 基础知识

中心扩展算法（解法一用到）

Manacher马拉车算法（解法二用到）

### 3 解法

#### 3.1 解法1（C++已过）

回文串有两种，一种长度为奇数，如aba，另一种长度为偶数，如abba，这两种的差别在于奇数的中心是一个字母，偶数的中心是两个字母，所以中心扩展算法就出来了。

遍历一遍所给字符串，然后分别以两种中心向外扩展回文串，在这过程就可以找到最长的回文串了

本人代码：

|  |
| --- |
| class Solution {  public:      //计算以left和right为中心向外扩展的回文串长度      int function(string s,int left,int right) {          int l = left,r = right;          while(l >= 0 && r < s.length() && s[l] == s[r]) {              l--;              r++;          }          return r - l - 1;      }      string longestPalindrome(string s) {          int start = 0,end = 0,len;          for(int i = 0; i < s.length(); i++) {              int len1 = function(s,i,i);//中心为一个字母的情况(回文串长度为奇数，如aba)              int len2 = function(s,i,i + 1);//中心为两个字母的情况(回文串长度为偶数，如abba)              len = max(len1,len2);//选择长的              if(len > end - start) {//更新始末坐标                  start = i - (len - 1) / 2;                  end = i + len / 2;              }          }          return s.substr(start,end - start + 1);//截取回文串并返回      }  }; |

#### 3.2 解法2（C++已过）

马拉车算法的模板题目。

关于此算法参考参考以下两个链接：

1. <https://blog.csdn.net/Form_/article/details/79766986>

2. <https://www.cnblogs.com/czsharecode/p/9705358.html>

本人代码：

|  |
| --- |
| class Solution {  public:      string Manacher(string &s)      {          //开头加'$'是为了防止越界          string t = "$#";          //在原字符串开头结尾以及字符之间插入一个字符(未在串中出现过的)          for(int i = 0; i < s.length(); i++)          {              t += s[i];              t += '#';          }          //p[i] - 1就是t串中以i为中点的回文长度          vector<int> p(t.length(),0);          //mx(某回文串延伸到的最右边下标),id(mx所属回文串中心下标)          int mx = 0,id = 0;          //bit（结果最大回文串中心下标）,maxlen（最大回文长度）          int bit,maxlen = 0;          for(int i = 1; i < t.length(); i++)          {              //算法核心              p[i] = mx > i ? min(p[2 \* id - i],mx - i) : 1;              //一个个进行比较              //t数组最左边 = '$',最右边 = '\0'，无需判断边界              while(t[i + p[i]] == t[i - p[i]])                  p[i]++;              //当t[i]匹配的 右边界超过mx时,mx和id就更新              if(i + p[i] > mx)              {                  mx = i + p[i];                  id = i;              }              //更新结果数据              if(p[i] > maxlen)              {                  maxlen = p[i];                  bit = i;              }          }          return s.substr((bit - maxlen) / 2,maxlen - 1);      }      string longestPalindrome(string s) {          return Manacher(s);      }  }; |

## [647. 回文子串](https://leetcode-cn.com/problems/palindromic-substrings/)

### 1 原题目

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

**示例 1：**

输入："abc"

输出：3

解释：三个回文子串: "a", "b", "c"

**示例 2：**

输入："aaa"

输出：6

解释：6个回文子串: "a", "a", "a", "aa", "aa", "aaa"

**提示：**

输入的字符串长度不会超过 1000 。

### 2 基础知识

Manacher马拉车算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

把马拉车算法多加三行代码就可以求解此问题。

本人代码：

|  |
| --- |
| class Solution {  public:      /\*      \* 求最长回文子串      \*/      int Manacher(string &s)//马拉车算法      {          //开头加'$'是为了防止越界          string t = "$#";          //在原字符串开头结尾以及字符之间插入一个字符(未在串中出现过的)          for(int i = 0; i < s.length(); i++)          {              t += s[i];              t += '#';          }          //p[i] - 1就是t串中以i为中点的回文长度          vector<int> p(t.length(),0);          //mx(某回文串延伸到的最右边下标),id(mx所属回文串中心下标)          int mx = 0,id = 0;          //bit（结果最大回文串中心下标）,maxlen（最大回文长度）          int bit,maxlen = 0;          int ans = 0;//新添加代码          for(int i = 1; i < t.length(); i++)          {              //算法核心              p[i] = mx > i ? min(p[2 \* id - i],mx - i) : 1;              //一个个进行比较              //t数组最左边 = '$',最右边 = '\0'，无需判断边界              while(t[i + p[i]] == t[i - p[i]])                  p[i]++;              //当t[i]匹配的 右边界超过mx时,mx和id就更新              if(i + p[i] > mx)              {                  mx = i + p[i];                  id = i;              }              //更新结果数据              if(p[i] > maxlen)              {                  maxlen = p[i];                  bit = i;              }              ans += (p[i] / 2);//新添加代码          }          //return s.substr((bit - maxlen) / 2,maxlen - 1);          //return maxlen - 1;          return ans;//新添加代码      }      int countSubstrings(string s) {          return Manacher(s);      }  }; |

## [685. 冗余连接 II](https://leetcode-cn.com/problems/redundant-connection-ii/)

### 1 原题目

在本问题中，有根树指满足以下条件的有向图。该树只有一个根节点，所有其他节点都是该根节点的后继。每一个节点只有一个父节点，除了根节点没有父节点。

输入一个有向图，该图由一个有着N个节点 (节点值不重复1, 2, ..., N) 的树及一条附加的边构成。附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。

结果图是一个以边组成的二维数组。 每一个边 的元素是一对 [u, v]，用以表示有向图中连接顶点 u 和顶点 v 的边，其中 u 是 v 的一个父节点。

返回一条能删除的边，使得剩下的图是有N个节点的有根树。若有多个答案，返回最后出现在给定二维数组的答案。

**示例 1:**

输入: [[1,2], [1,3], [2,3]]

输出: [2,3]

解释: 给定的有向图如下:

1

/ \

v v

2-->3

**示例 2:**

输入: [[1,2], [2,3], [3,4], [4,1], [1,5]]

输出: [4,1]

解释: 给定的有向图如下:

5 <- 1 -> 2

^ |

| v

4 <- 3

**注意:**

二维数组大小的在3到1000范围内。

二维数组中的每个整数在1到N之间，其中 N 是二维数组的大小。

### 2 基础知识

并查集（解法一用到）

图（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题一开始我做的时候想到了要从图的入度来做，可惜没解出来。。。。实际上此题考查的是你并查集的知识。你说牛掰不牛掰。

题目是N个点的有根树，那么此树就是n - 1条边，所以每个点的入度都要为1，根节点入度为0。

1. 第一种就是判断每个点的入度，入度为2的点一定有一条边是附加边。
2. 第二种就是所有点入度为1，那么就成环了，直接从后往前一个个的试就可以了。

判断此边是不是附加边用并查集就可以了，如果是附加边的话，删除了此边那么树的N个点还是会在一个集合中。如果不是附加边，那么删除了就会使树分成两个。

参考链接：https://leetcode-cn.com/problems/redundant-connection-ii/solution/cjian-hua-guan-fang-si-lu-bing-cha-ji-by-energy-7/

本人代码：

|  |
| --- |
| class Solution {  public:      #define M 1005      int father[M];//记录父节点      int Find(int x)//查询x的根节点并路径压缩      {          if(father[x] != x)              father[x] = Find(father[x]);          return father[x];      }      void Union(int x,int y)//合并x和y的集合      {          int rx,ry;          rx=Find(x);          ry=Find(y);          if(rx!=ry)              father[rx]=ry;      }      bool Query(int a,int b)//查询a和b是否在同一集合      {          return Find(a) == Find(b);//二者根节点相同即为在同一集合      }      //判断删除edges[n][0]和edges[n][1]连接的边会不会使图分成两部分      //如果不会分成两部分那说明此边就是附加边（题目要求是使得剩下的图是有N个节点的有根树）      bool samefather(vector<vector<int>>& edges,int n) {          for (int i = 0; i < M; i++)              father[i] = i;          for (int i = 0; i < edges.size(); i++) {              if (i == n)                  continue;              Union(edges[i][0],edges[i][1]);          }          if (Query(edges[n][0],edges[n][1]))              return true;          return false;      }      vector<int> findRedundantDirectedConnection(vector<vector<int>>& edges) {          vector<vector<int>> in(edges.size() + 1);//入度          //1.某个点如果入度为2，那么入此点的两条边中一定有一个是附加边          for (int i = 0; i < edges.size(); i++) {              in[edges[i][1]].push\_back(i);              if (in[edges[i][1]].size() == 2) {                  if (samefather(edges,in[edges[i][1]][1]))                      return {edges[in[edges[i][1]][1]][0],edges[in[edges[i][1]][1]][1]};                  else                      return {edges[in[edges[i][1]][0]][0],edges[in[edges[i][1]][0]][1]};              }          }          //2.所有点入度都为1，就成环了，所以就暴力从后往前找（题目要求若有多个答案，返回最后出现在给定二维数组的答案。）          for (int i = edges.size() - 1; i >= 0; i--){              if (samefather(edges,i))                  return {edges[i][0],edges[i][1]};          }          return {};      }  }; |

## [990. 等式方程的可满足性](https://leetcode-cn.com/problems/satisfiability-of-equality-equations/)

### 1 原题目

给定一个由表示变量之间关系的字符串方程组成的数组，每个字符串方程 equations[i] 的长度为 4，并采用两种不同的形式之一："a==b" 或 "a!=b"。在这里，a 和 b 是小写字母（不一定不同），表示单字母变量名。

只有当可以将整数分配给变量名，以便满足所有给定的方程时才返回 true，否则返回 false。

**示例 1：**

输入：["a==b","b!=a"]

输出：false

解释：如果我们指定，a = 1 且 b = 1，那么可以满足第一个方程，但无法满足第二个方程。没有办法分配变量同时满足这两个方程。

**示例 2：**

输出：["b==a","a==b"]

输入：true

解释：我们可以指定 a = 1 且 b = 1 以满足满足这两个方程。

**示例 3：**

输入：["a==b","b==c","a==c"]

输出：true

**示例 4：**

输入：["a==b","b!=c","c==a"]

输出：false

**示例 5：**

输入：["c==c","b==d","x!=z"]

输出：true

**提示：**

1 <= equations.length <= 500

equations[i].length == 4

equations[i][0] 和 equations[i][3] 是小写字母

equations[i][1] 要么是 '='，要么是 '!'

equations[i][2] 是 '='

### 2 基础知识

并查集（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

本题可以利用并查集的知识来解决，将所有相等的分到各自的集合中，然后在对不等的进行判断，不能的两个字母不能在同一个集合中。

本人代码：

|  |
| --- |
| class Solution {  public:      int father[1000];//记录父节点      int Find(int x)//查询x的根节点并路径压缩      {          if(father[x] != x)              father[x] = Find(father[x]);          return father[x];      }      void Union(int x,int y)//合并x和y的集合      {          int rx,ry;          rx=Find(x);          ry=Find(y);          if(rx!=ry)              father[rx]=ry;      }      bool Query(int a,int b)//查询a和b是否在同一集合      {          return Find(a) == Find(b);//二者根节点相同即为在同一集合      }      bool equationsPossible(vector<string>& equations) {          int i;          string temp;          for(i = 0; i < 1000; i++)//初始化              father[i] = i;          for(i = 0; i < equations.size(); i++) {              temp = equations[i];              if(temp[1] == '=')                  Union(temp[0] - 26,temp[3] - 26);//合并          }          for(i = 0; i < equations.size(); i++) {              temp = equations[i];              if(temp[1] == '!') {                  if(Query(temp[0] - 26,temp[3] - 26))//查询                      return false;              }          }          return true;      }  }; |

## [739. 每日温度](https://leetcode-cn.com/problems/daily-temperatures/)

### 1 原题目

根据每日 气温 列表，请重新生成一个列表，对应位置的输出是需要再等待多久温度才会升高超过该日的天数。如果之后都不会升高，请在该位置用 0 来代替。

例如，给定一个列表 temperatures = [73, 74, 75, 71, 69, 72, 76, 73]，你的输出应该是 [1, 1, 4, 2, 1, 1, 0, 0]。

**提示：**气温 列表长度的范围是 [1, 30000]。每个气温的值的均为华氏度，都是在 [30, 100] 范围内的整数。

### 2 基础知识

单调栈（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

本题的做法是维护一个单调栈来查找结果（栈内元素始终是栈顶到栈底呈递减）。每个温度都是在它后面的天数找寻比他高的温度。本题栈里面存的是温度对应的下标

最后天数找不到所以是0；

我们从后往前遍历整个数组，将当前温度和栈顶温度比较，如果当前遍历到的温度大于栈顶温度，那么就移除栈顶元素，继续取栈顶元素，直到栈为空（当前温度后面天数没有比它高的，结果就是0，并将当前温度下标入栈）或者找到栈内温度大于当前遍历的温度（下标相减就是结果，并将当前温度下标入栈）

本人代码：

|  |
| --- |
| class Solution {  public:      vector<int> dailyTemperatures(vector<int>& T) {          stack<int>s;//栈          vector<int> res(T.size(),0);//结果          int temp = T.size() - 1;          for(int i = T.size() - 1; i >= 0; i--) {              if(!s.empty())                  temp = s.top();              while(!s.empty() && T[temp] <= T[i]) {//跟栈顶元素比较                  s.pop();                  if(!s.empty())                      temp = s.top();              }              if(T[temp] > T[i]) {                  res[i] = temp - i;              }              s.push(i);          }          return res;      }  }; |

## [215. 数组中的第K个最大元素](https://leetcode-cn.com/problems/kth-largest-element-in-an-array/)

### 1 原题目

在未排序的数组中找到第 k 个最大的元素。请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

**示例 1:**

输入: [3,2,1,5,6,4] 和 k = 2

输出: 5

**示例 2:**

输入: [3,2,3,1,2,4,5,5,6] 和 k = 4

输出: 4

**说明:**

你可以假设 k 总是有效的，且 1 ≤ k ≤ 数组的长度。

### 2 基础知识

堆排序（解法一用到）

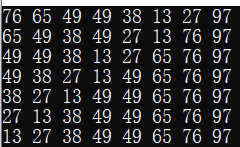
### 3 解法

#### 3.1 解法1（C++已过）

利用堆排序来找到数组中第K个最大的元素。

代码中的高亮标记的for循环很重要，正常堆排序的for循环应该是for(int i = numSize - 1; i > 0; i--)

假如有这个一样的数组{49,38,65,97,76,13,27,49}



上图是for循环中i>0时的运行结果。

所以我们发现每次对堆调整的时候数组的头元素就是下一个最大的元素，所以此题找第K大元素，所以我们在代码for循环i结束时加上限制条件就可以（就是限制调整的次数）。

本人代码：

|  |
| --- |
| //堆排序  void HeapAdjust(int \*num,int s,int m)//筛选法调整堆  {      //假设num[s+1..m]已经是堆，将num[s..m]调整为以num[s]为根的大根堆      int temp = num[s];      for(int i = 2 \* s; i <= m; i \*= 2)//沿值较大的孩子结点向下筛选      {          if(i < m && num[i] < num[i + 1])              i++;//i为较大值的下标          if(temp >= num[i])//temp应插在位置s上              break;          num[s] = num[i];          s = i;      }      num[s] = temp;//插入  }  void CreatHeap(int \*num,int numSize)//建初堆  {      //把无序数组num[0..n]建成大根堆      int n = numSize - 1;      for(int i = n / 2; i >= 0; i--)//反复调用HeapAdjust          HeapAdjust(num,i,n);  }  void HeapSort(int \*num,int numSize,int k)//堆排序  {      //对数组num进行堆排序      CreatHeap(num,numSize);//把无序数组num[0..num.size() - 1]建成大根堆      for(int i = numSize - 1; i >= numSize - k + 1; i--)      {          int temp = num[0];//将堆顶记录和当前未经排序子数组num[0..i]中最后一个记录互换          num[0] = num[i];          num[i] = temp;          HeapAdjust(num,0,i - 1);//将num[0..i-1]重新调整为大根堆      }  }  int findKthLargest(int\* nums, int numsSize, int k){      HeapSort(nums,numsSize,k);      return nums[0];  } |

## [347. 前 K 个高频元素](https://leetcode-cn.com/problems/top-k-frequent-elements/)

### 1 原题目

给定一个非空的整数数组，返回其中出现频率前 k 高的元素。

**示例 1:**

输入: nums = [1,1,1,2,2,3], k = 2

输出: [1,2]

**示例 2:**

输入: nums = [1], k = 1

输出: [1]

**提示：**

你可以假设给定的 k 总是合理的，且 1 ≤ k ≤ 数组中不相同的元素的个数。

你的算法的时间复杂度必须优于 O(n log n) , n 是数组的大小。

题目数据保证答案唯一，换句话说，数组中前 k 个高频元素的集合是唯一的。

你可以按任意顺序返回答案。

### 2 基础知识

堆排序（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题没什么难的，就是利用堆排序来找出出现频率前k高的元素。堆排序我们又借助了优先队列，所以只需自己构造一个小顶堆的比较函数就可以了（pair类型的），然后就解出来了。

本人代码：

|  |
| --- |
| class Solution {  public:      //小顶堆      struct cmp {          bool operator() (const pair<int, int>& lhs, const pair<int, int>& rhs) {              return lhs.second > rhs.second;          }      };      vector<int> topKFrequent(vector<int>& nums, int k) {          //记录元素出现次数          unordered\_map<int,int> num;          for (int i = 0; i < nums.size(); i++) {              num[nums[i]]++;          }          //利用小顶堆来比较出来出现频率前k高的元素          priority\_queue<pair<int,int>,vector<pair<int,int>>,cmp> que;          for (unordered\_map<int,int>::iterator it = num.begin(); it != num.end(); it++) {              que.push(\*it);              if (que.size() > k) {                  que.pop();              }          }          //因为是小顶堆所以倒着存储          vector<int> res(k);          for (int i = k - 1; i >= 0; i--) {              res[i] = que.top().first;              que.pop();          }          return res;      }  }; |

## [面试题 17.13. 恢复空格](https://leetcode-cn.com/problems/re-space-lcci/)

### 1 原题目

哦，不！你不小心把一个长篇文章中的空格、标点都删掉了，并且大写也弄成了小写。像句子"I reset the computer. It still didn’t boot!"已经变成了"iresetthecomputeritstilldidntboot"。在处理标点符号和大小写之前，你得先把它断成词语。当然了，你有一本厚厚的词典dictionary，不过，有些词没在词典里。假设文章用sentence表示，设计一个算法，把文章断开，要求未识别的字符最少，返回未识别的字符数。

**注意：**本题相对原题稍作改动，只需返回未识别的字符数

**示例：**

输入：

dictionary = ["looked","just","like","her","brother"]

sentence = "jesslookedjustliketimherbrother"

输出： 7

解释： 断句后为"jess looked just like tim her brother"，共7个未识别字符。

**提示：**

0 <= len(sentence) <= 1000

dictionary中总字符数不超过 150000。

你可以认为dictionary和sentence中只包含小写字母。

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

暴力dp，dp[i]代表前i个字符[0,i)中最少未识别字符，如果当前字符不能和前面的字符组成单词，那肯定转移方程就是 dp[i] = dp[i - 1] + 1。如果以当前字符结尾能组成长度为len的字典中的单词，那么dp[i] = dp[i – len]（当然，也可能组成好多个字典中的单词，所以这个方程我们要取最小值）。转移方程推到完毕，就剩下初值问题，初值就是dp[0] = 0，前0个字符肯定没有未识别字符。

本人代码：

|  |
| --- |
| class Solution {  public:      int respace(vector<string>& dictionary, string sentence) {          vector<int> len;//记录字典中所有单词的长度          unordered\_set<int> l;//标记字典中单词的长度          unordered\_set<string> s;//标记字典中出现的单词          for(int i = 0; i < dictionary.size(); i++) {              s.insert(dictionary[i]);              if(l.find(dictionary[i].length()) == l.end()) {                  len.push\_back(dictionary[i].length());                  l.insert(dictionary[i].length());              }          }          sort(len.begin(),len.end());//从小到大排序          vector<int> dp(sentence.length() + 1,0);          //暴力dp，dp[i]代表前i个字符[0,i)中最少未识别字符          for(int i = 1; i <= sentence.length(); i++) {              //先假设当前字母和前面的字母无法组成字典中的单词              dp[i] = dp[i - 1] + 1;//未识别字符加1              //当前字母能否和前面的字符组成字典中的单词              for(int j = 0; j < len.size(); j++) {                  if(len[j] > i)                      break;                  //组成字典中的单词的情况                  if(s.find(sentence.substr(i - len[j],len[j])) != s.end()) {                      dp[i] = min(dp[i],dp[i - len[j]]);//取最小值                  }              }          }          return dp[sentence.length()];      }  }; |

## [1503. 所有蚂蚁掉下来前的最后一刻](https://leetcode-cn.com/problems/last-moment-before-all-ants-fall-out-of-a-plank/)

### 1 原题目

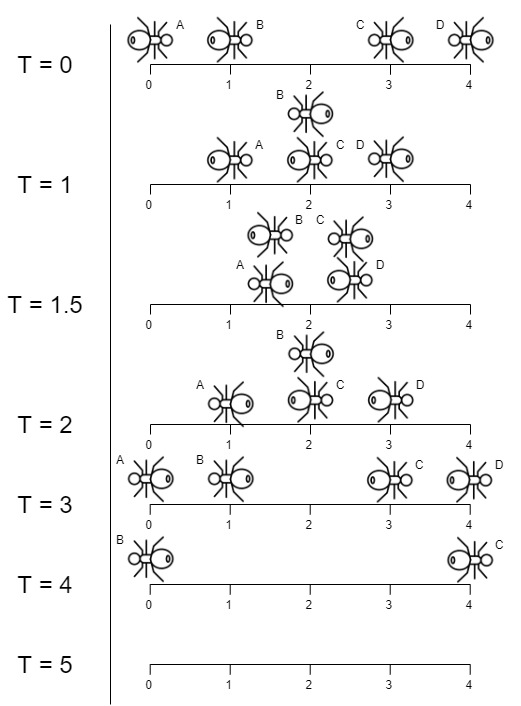
有一块木板，长度为 n 个 单位 。一些蚂蚁在木板上移动，每只蚂蚁都以 每秒一个单位 的速度移动。其中，一部分蚂蚁向 左 移动，其他蚂蚁向 右 移动。

当两只向 不同 方向移动的蚂蚁在某个点相遇时，它们会同时改变移动方向并继续移动。假设更改方向不会花费任何额外时间。

而当蚂蚁在某一时刻 t 到达木板的一端时，它立即从木板上掉下来。

给你一个整数 n 和两个整数数组 left 以及 right 。两个数组分别标识向左或者向右移动的蚂蚁在 t = 0 时的位置。请你返回最后一只蚂蚁从木板上掉下来的时刻。

**示例 1：**



输入：n = 4, left = [4,3], right = [0,1]

输出：4

解释：如上图所示：

-下标 0 处的蚂蚁命名为 A 并向右移动。

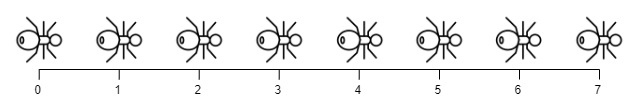
-下标 1 处的蚂蚁命名为 B 并向右移动。

-下标 3 处的蚂蚁命名为 C 并向左移动。

-下标 4 处的蚂蚁命名为 D 并向左移动。

请注意，蚂蚁在木板上的最后时刻是 t = 4 秒，之后蚂蚁立即从木板上掉下来。（也就是说在 t = 4.0000000001 时，木板上没有蚂蚁）。

**示例 2：**

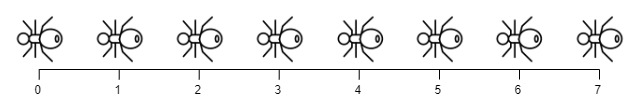


输入：n = 7, left = [], right = [0,1,2,3,4,5,6,7]

输出：7

解释：所有蚂蚁都向右移动，下标为 0 的蚂蚁需要 7 秒才能从木板上掉落。

**示例 3：**



输入：n = 7, left = [0,1,2,3,4,5,6,7], right = []

输出：7

解释：所有蚂蚁都向左移动，下标为 7 的蚂蚁需要 7 秒才能从木板上掉落。

**示例 4：**

输入：n = 9, left = [5], right = [4]

输出：5

解释：t = 1 秒时，两只蚂蚁将回到初始位置，但移动方向与之前相反。

**示例 5：**

输入：n = 6, left = [6], right = [0]

输出：6

**提示：**

1 <= n <= 10^4

0 <= left.length <= n + 1

0 <= left[i] <= n

0 <= right.length <= n + 1

0 <= right[i] <= n

1 <= left.length + right.length <= n + 1

left 和 right 中的所有值都是唯一的，并且每个值 只能出现在二者之一 中。

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

蚂蚁掉头其实你就可以看做成蚂蚁直接穿过遇见到的蚂蚁（各走各的，不用转向）。所以问题就可以变成直接找蚂蚁中落地最晚（不用管转不转向的问题）。

本人代码：

|  |
| --- |
| class Solution {  public:      int getLastMoment(int n, vector<int>& left, vector<int>& right) {          int res = 0;          //找向左爬的蚂蚁中落地最晚的          for(int i = 0; i < left.size(); i++)              res = max(res,left[i]);          //找向右爬的蚂蚁中落地最晚的          for(int i = 0; i < right.size(); i++)              res = max(res,n - right[i]);          return res;      }  }; |

## [122. 买卖股票的最佳时机 II](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/)

### 1 原题目

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

**示例 1:**

输入: [7,1,5,3,6,4]

输出: 7

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出, 这笔交易所能获得利润 = 5-1 = 4 。

  随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出, 这笔交易所能获得利润 = 6-3 = 3 。

**示例 2:**

输入: [1,2,3,4,5]

输出: 4

解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出, 这笔交易所能获得利润 = 5-1 = 4 。

  注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。

  因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

**示例 3:**

输入: [7,6,4,3,1]

输出: 0

解释: 在这种情况下, 没有交易完成, 所以最大利润为 0。

**提示：**

1 <= prices.length <= 3 \* 10 ^ 4

0 <= prices[i] <= 10 ^ 4

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

题目其实告诉了我们一个隐藏的信息，那就是今天卖了股票后，今天还可以购入股票。

知道这个信息，那么此题就变得简单了。对于连续下降的，我们肯定不会买，买了就亏损，而对于上涨的和连续上涨的，我们肯定要买，这样有利润。单独的一个上涨的，很好办，就是后一天减去今天就是利润，那么连续上涨的呢？这就要用到了题目的那个隐藏信息，我们也可以这样一天一天的减然后加起来，和连续上涨的最高点减最低点的值是一样的。所以我们可以遍历股票价格，相邻两天如果有利润，就计算出来加到结果中。

本人代码：

|  |
| --- |
| class Solution {  public:      int maxProfit(vector<int>& prices) {          int res = 0;          //相邻两天买卖能获得利润的都加到一起去          for(int i = 1; i < prices.size(); i++)              if(prices[i] > prices[i - 1])                  res += prices[i] - prices[i - 1];          return res;      }  }; |

## [315. 计算右侧小于当前元素的个数](https://leetcode-cn.com/problems/count-of-smaller-numbers-after-self/)

### 1 原题目

给定一个整数数组 nums，按要求返回一个新数组 counts。数组 counts 有该性质： counts[i] 的值是  nums[i] 右侧小于 nums[i] 的元素的数量。

**示例：**

输入：[5,2,6,1]

输出：[2,1,1,0]

解释：

5 的右侧有 2 个更小的元素 (2 和 1)

2 的右侧仅有 1 个更小的元素 (1)

6 的右侧有 1 个更小的元素 (1)

1 的右侧有 0 个更小的元素

### 2 基础知识

树状数组（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

本题可以利用树状数组来求解，因为树状数组本身就是用来求前缀和的，而本题让你求右侧小于当前元素的个数，所以我们从后往前的建立树状数组，这样就可以利用前缀和来求解此题，假如数组为[3,2,2,6,6,9,2,5,4]，我们从后往前，一个个将元素加到树状数组中，当遍历到9时，9的前缀和就是c[1]到c[8]自然就是9右侧小于9的数量（因为倒着加进去的），但是有两点需要注意的，此题树状数组更新值加的值固定为1（毕竟此题是统计个数），还有一点需要注意的是树状数组空间消耗很大，但此题给的数据可能很稀疏，所以我们需要将数组进行离散化处理来节省空间。

本人代码：

|  |
| --- |
| class Solution {  public:      vector<int> c;//树状数组      vector<int> a;//nums离散化处理后成a数组      void Init(int len) {          //对树状数组设置大小          c.resize(len, 0);      }      void Discretization(vector<int>& nums) {          //拷贝数组          a.assign(nums.begin(), nums.end());          //对数组进行排序          sort(a.begin(), a.end());          //对数组进行去重          a.erase(unique(a.begin(), a.end()), a.end());      }      int getId(int x) {          //lower\_bound()返回值是一个迭代器,返回指向大于等于x的第一个值的位置          //lower\_bound()函数内部就是二分查找          //加一的目的是因为树状数组是从下标1的位置开始存储          return lower\_bound(a.begin(), a.end(), x) - a.begin() + 1;      }      //下面这三个函数就是树状数组-单点更新区间查询的模板      int lowbit(int x) {          return x&(-x);      }      int getsum(int i) {//求a[1]到a[i]的和(包含a[i])          int res = 0;          while(i > 0)          {              res += c[i];              i -= lowbit(i);          }          return res;      }      void updata(int i) {//在i位置加上k(此题k固定为1)（此函数只更改树状数组的值，若要更改原数组的值需要在主函数中实现）          while(i < c.size())          {              c[i] += 1;              i += lowbit(i);          }      }      vector<int> countSmaller(vector<int>& nums) {          vector<int> res;          Init(nums.size() + 5);          Discretization(nums);          for(int i = nums.size() - 1; i >= 0; --i) {              int id = getId(nums[i]);              updata(id);              res.push\_back(getsum(id - 1));          }          reverse(res.begin(), res.end());//翻转          return res;      }  }; |

## [1025. 除数博弈](https://leetcode-cn.com/problems/divisor-game/)

### 1 原题目

爱丽丝和鲍勃一起玩游戏，他们轮流行动。爱丽丝先手开局。

最初，黑板上有一个数字 N 。在每个玩家的回合，玩家需要执行以下操作：

选出任一 x，满足 0 < x < N 且 N % x == 0 。

用 N - x 替换黑板上的数字 N 。

如果玩家无法执行这些操作，就会输掉游戏。

只有在爱丽丝在游戏中取得胜利时才返回 True，否则返回 false。假设两个玩家都以最佳状态参与游戏。

**示例 1：**

输入：2

输出：true

解释：爱丽丝选择 1，鲍勃无法进行操作。

**示例 2：**

输入：3

输出：false

解释：爱丽丝选择 1，鲍勃也选择 1，然后爱丽丝无法进行操作。

**提示：**

1 <= N <= 1000

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

本人刚开始做本题用的是赖皮方法，N才小于等于1000，所以我直接先把所有奇数打进样例中，再把所有偶数打进样例中，就发现此题做出来了，啊哈哈哈哈哈，这是赖皮解法。

正常应该是按照题解进行推理的，参考链接：<https://leetcode-cn.com/problems/divisor-game/solution/chu-shu-bo-yi-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      bool divisorGame(int N) {          //N 为奇数的时候 Alice（先手）必败，N 为偶数的时候 Alice 必胜          if(N % 2 == 0)              return true;          return false;      }  }; |

## [1514. 概率最大的路径](https://leetcode-cn.com/problems/path-with-maximum-probability/)

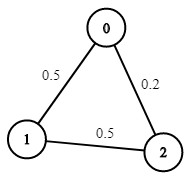
### 1 原题目

给你一个由 n 个节点（下标从 0 开始）组成的无向加权图，该图由一个描述边的列表组成，其中 edges[i] = [a, b] 表示连接节点 a 和 b 的一条无向边，且该边遍历成功的概率为 succProb[i] 。

指定两个节点分别作为起点 start 和终点 end ，请你找出从起点到终点成功概率最大的路径，并返回其成功概率。

如果不存在从 start 到 end 的路径，请 返回 0 。只要答案与标准答案的误差不超过 1e-5 ，就会被视作正确答案。

**示例 1：**

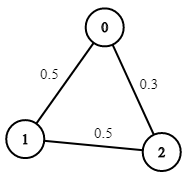


输入：n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2

输出：0.25000

解释：从起点到终点有两条路径，其中一条的成功概率为 0.2 ，而另一条为 0.5 \* 0.5 = 0.25

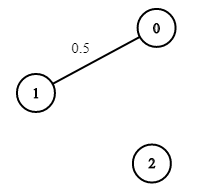
**示例 2：**



输入：n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.3], start = 0, end = 2

输出：0.30000

**示例 3：**



输入：n = 3, edges = [[0,1]], succProb = [0.5], start = 0, end = 2

输出：0.00000

解释：节点 0 和 节点 2 之间不存在路径

**提示：**

2 <= n <= 10^4

0 <= start, end < n

start != end

0 <= a, b < n

a != b

0 <= succProb.length == edges.length <= 2\*10^4

0 <= succProb[i] <= 1

每两个节点之间最多有一条边

### 2 基础知识

SPFA算法（解法一用到）

迪杰斯特拉算法 + 堆优化（解法二用到）

### 3 解法

#### 3.1 解法1（C++已过）

这种解法就是SPFA算法的模板直接套上去就可以，需要注意的是概率和最短路不同，概率我们找的是最大值，并且源点到自身的概率为1。

本人代码：

|  |
| --- |
| class Solution {  public:      #define MAXN 40005      #define N 10005      struct Edge//边结构体      {          int next;//与第i条边同一起点的上一条边的位置          int to;//此边的终点          double len;//边的权值      } edge[MAXN];      int head[N];//以i为起点的第一条边存储的位置（实际是最后输入的那个编号）      int index;//若有多组例子，要在主函数里对它进行重新置为零的初始化      bool vis[N];//用来标记点是否在队列中      double length[N];//记录从起点到点n的路径的概率      void add(int start,int end,double len)//链式前向星的加边函数      {          edge[index].len = len;          edge[index].to = end;          edge[index].next = head[start];          head[start] = index++;      }      //最短路SPFA算法      void SPFA(int start,int n) {//start是起始节点，n是节点总数          memset(vis,0,sizeof(vis));          memset(length,0,sizeof(length));          queue<int>que;          que.push(start);//入队          vis[start] = true;//标记此点入队列          length[start] = 1;//源点到源点的概率为1          while(!que.empty()) {              int u = que.front();              que.pop();//出队              vis[u] = false;//标记此点已经不在队列中              for(int j = head[u]; j >= 0; j = edge[j].next) {//遍历与u联通的点                  int v = edge[j].to;                  if(length[v] < length[u] \* edge[j].len) {                      length[v] = length[u] \* edge[j].len;//路径概率更新                      if(vis[v] == 0) {//如果v点不在队列中                          que.push(v);//入队                          vis[v] = 1;//标记                      }                  }              }          }      }      double maxProbability(int n, vector<vector<int>>& edges, vector<double>& succProb, int start, int end) {          memset(head,-1,sizeof(head));          index = 0;          for(int i = 0; i < edges.size(); i++) {              add(edges[i][0],edges[i][1],succProb[i]);              add(edges[i][1],edges[i][0],succProb[i]);          }          SPFA(start,n);//n为有向图顶点数，0为源点（由题意来输入源点,本代码点的编号是[0,n)）          return length[end];      }  }; |

#### 3.2 解法2（C++已过）

此题这种解法如果没用堆优化就会超时，所以需要将迪杰斯特拉算法进行堆优化就可以通过。堆优化的迪杰斯特拉算法其实就是利用了优先队列的性质，不用像初始版本的迪杰斯特拉算法那样for循环进行寻找点，优先队列的性质其实内部就是一个大根堆，所以在最短路问题中需要重写排序函数，但是吧，此题目是求得概率最大，所以可以不用重写排序函数（我是直接套的板子，所以排序函数也有）。也算是一道板子题。

本人代码：

|  |
| --- |
| class Solution {  public:      #define MAXN 40005      #define N 10005      struct qnode//根据c值的大小，c值大的先出队列      {          int v;          double c;          qnode(int \_v = 0,double \_c = 0):v(\_v),c(\_c) {}          bool operator <(const qnode &r)const          {              return c < r.c;          }      };      struct Edge      {          int v;//终点          double cost;//起点到终点v遍历成功的概率（图中的信息）          Edge(int \_v = 0,double \_cost = 0):v(\_v),cost(\_cost) {}      };      vector<Edge> E[MAXN];//存储图的信息      bool vis[MAXN];//标记数组      double dist[MAXN];//记录起始点到当前下标的概率      //Dijkstar算法 + 堆优化      void Dijkstra(int n,int start)//点的编号从0开始      {          //初始化          memset(vis,false,sizeof(vis));          for(int i = 0; i < n; i++)              dist[i] = 0;          priority\_queue<qnode> que;//优先队列          while(!que.empty())              que.pop();          dist[start] = 1;//起点到起点的概率为1          que.push(qnode(start,1));          qnode tmp;          while(!que.empty())          {              tmp = que.top();              que.pop();              int u = tmp.v;              if(vis[u])                  continue;              vis[u] = true;              //遍历与之联通的点              for(int i = 0; i < E[u].size(); i++)              {                  int v = E[tmp.v][i].v;                  double cost = E[u][i].cost;                  if(!vis[v] && dist[v] < dist[u] \* cost)                  {                      dist[v] = dist[u] \* cost;//路径更新                      que.push(qnode(v,dist[v]));                  }              }          }      }      //建图      void addedge(int u,int v,double w)      {          E[u].push\_back(Edge(v,w));      }      double maxProbability(int n, vector<vector<int>>& edges, vector<double>& succProb, int start, int end) {          for(int i = 0; i < edges.size(); i++) {              addedge(edges[i][0],edges[i][1],succProb[i]);              addedge(edges[i][1],edges[i][0],succProb[i]);          }          //n是点的个数，start是起点          Dijkstra(n,start);          return dist[end];      }  }; |

## [343. 整数拆分](https://leetcode-cn.com/problems/integer-break/)

### 1 原题目

给定一个正整数 n，将其拆分为至少两个正整数的和，并使这些整数的乘积最大化。 返回你可以获得的最大乘积。

**示例 1:**

输入: 2

输出: 1

解释: 2 = 1 + 1, 1 × 1 = 1。

**示例 2:**

输入: 10

输出: 36

解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36。

**说明:** 你可以假设 n 不小于 2 且不大于 58。

### 2 基础知识

数学（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此解法吧，我看了下LeetCode下面的提示，发现：

2 = 1 + 1 8 = 3 + 3 + 2

3 = 1 + 3 9 = 3 + 3 + 3

4 = 2 + 2 10 = 3 + 3 + 4

5 = 3 + 2 11 = 3 + 3 + 3 + 2

6 = 3 + 3 12 = 3 + 3 + 3 + 3

7 = 3 + 4 13 = 3 + 3 + 3 + 4

这些数字只有化成这些数字相加乘积才最大，所以貌似可以发现点规律，就是从数字5以后，都是尽可能多的先取3，直到剩下2，3，4中任意一个数字。所以就可以编写出下面的代码。

我是发现的规律，实际上这个是数学证明的，参考链接：<https://leetcode-cn.com/problems/integer-break/solution/zheng-shu-chai-fen-by-leetcode-solution/>

本人代码：

|  |
| --- |
| class Solution {  public:      int integerBreak(int n) {          if(n <= 3)              return n - 1;          int res = 1;          while(n > 4) {              res \*= 3;              n -=3 ;          }          res \*= n;          return res;      }  }; |

## [1529. 灯泡开关 IV](https://leetcode-cn.com/problems/bulb-switcher-iv/)

### 1 原题目

房间中有 n 个灯泡，编号从 0 到 n-1 ，自左向右排成一行。最开始的时候，所有的灯泡都是关着的。

请你设法使得灯泡的开关状态和 target 描述的状态一致，其中 target[i] 等于 1 第 i 个灯泡是开着的，等于 0 意味着第 i 个灯是关着的。

有一个开关可以用于翻转灯泡的状态，翻转操作定义如下：

选择当前配置下的任意一个灯泡（下标为 i ）

翻转下标从 i 到 n-1 的每个灯泡

翻转时，如果灯泡的状态为 0 就变为 1，为 1 就变为 0 。

返回达成 target 描述的状态所需的 最少 翻转次数。

**示例 1：**

输入：target = "10111"

输出：3

解释：初始配置 "00000".

从第 3 个灯泡（下标为 2）开始翻转 "00000" -> "00111"

从第 1 个灯泡（下标为 0）开始翻转 "00111" -> "11000"

从第 2 个灯泡（下标为 1）开始翻转 "11000" -> "10111"

至少需要翻转 3 次才能达成 target 描述的状态

**示例 2：**

输入：target = "101"

输出：3

解释："000" -> "111" -> "100" -> "101".

**示例 3：**

输入：target = "00000"

输出：0

**示例 4：**

输入：target = "001011101"

输出：5

**提示：**

1 <= target.length <= 10^5

target[i] == '0' 或者 target[i] == '1'

### 2 基础知识

智力数学（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

刚开始得做法是依此翻转，但是超时了，我先说一下我第一次的思路。

比如： 10111，初始是00000

翻转一次：01000

翻转两次：00111

翻转三次：00000

这种思路就是每次解决最前面的一个1，没毛病吧，结果超时。

突然发现，其实我的那种翻转就是统计相邻字符是否相同，不同就计数加一。首字母要额外判断，是1就要额外多翻转一次。

本人代码：

|  |
| --- |
| class Solution {  public:      int minFlips(string target) {          //比较相邻字符是否相同，不同就计数加一。另外首位的字母也要判断是否为'1'（要进行额外的翻转）          int step = 0;          if(target[0] == '1')              step++;          for(int i = 1; i < target.length(); i++) {              if(target[i] != target[i - 1])                  step++;          }          return step;      }  }; |

## [133. 克隆图](https://leetcode-cn.com/problems/clone-graph/)

### 1 原题目

给你无向 连通 图中一个节点的引用，请你返回该图的 深拷贝（克隆）。

图中的每个节点都包含它的值 val（int） 和其邻居的列表（list[Node]）。

class Node {

public int val;

public List<Node> neighbors;

}

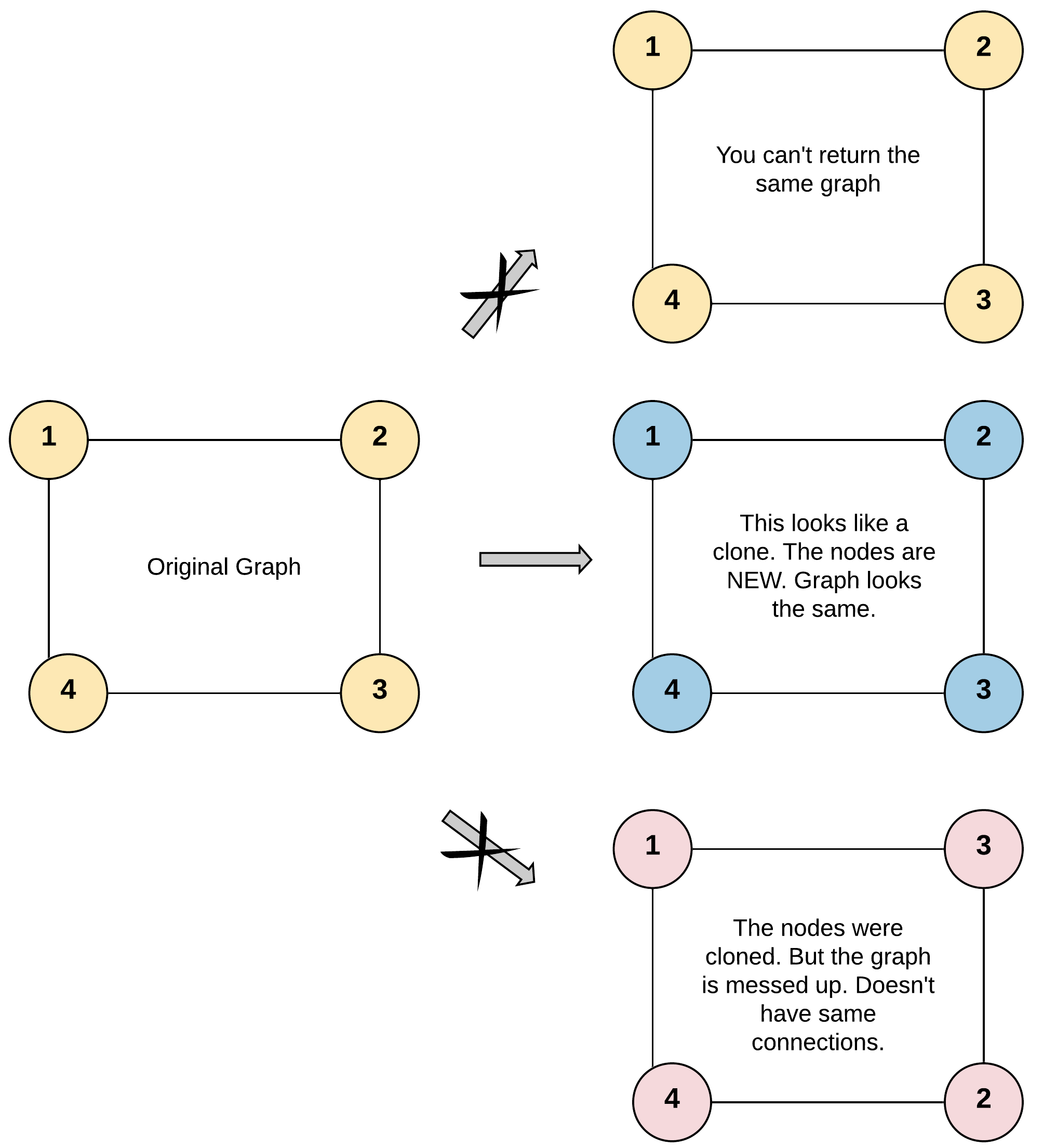
**测试用例格式：**

简单起见，每个节点的值都和它的索引相同。例如，第一个节点值为 1（val = 1），第二个节点值为 2（val = 2），以此类推。该图在测试用例中使用邻接列表表示。

邻接列表 是用于表示有限图的无序列表的集合。每个列表都描述了图中节点的邻居集。

给定节点将始终是图中的第一个节点（值为 1）。你必须将 给定节点的拷贝 作为对克隆图的引用返回。

**示例 1：**



输入：adjList = [[2,4],[1,3],[2,4],[1,3]]

输出：[[2,4],[1,3],[2,4],[1,3]]

解释：

图中有 4 个节点。

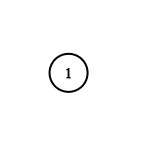
节点 1 的值是 1，它有两个邻居：节点 2 和 4 。

节点 2 的值是 2，它有两个邻居：节点 1 和 3 。

节点 3 的值是 3，它有两个邻居：节点 2 和 4 。

节点 4 的值是 4，它有两个邻居：节点 1 和 3 。

**示例 2：**



输入：adjList = [[]]

输出：[[]]

解释：输入包含一个空列表。该图仅仅只有一个值为 1 的节点，它没有任何邻居。

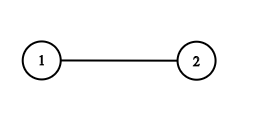
**示例 3：**

输入：adjList = []

输出：[]

解释：这个图是空的，它不含任何节点。

**示例 4：**



输入：adjList = [[2],[1]]

输出：[[2],[1]]

**提示：**

节点数不超过 100 。

每个节点值 Node.val 都是唯一的，1 <= Node.val <= 100。

无向图是一个简单图，这意味着图中没有重复的边，也没有自环。

由于图是无向的，如果节点 p 是节点 q 的邻居，那么节点 q 也必须是节点 p 的邻居。

图是连通图，你可以从给定节点访问到所有节点。

### 2 基础知识

广度优先搜索（解法一用到）

深度优先搜索（解法二用到）

### 3 解法

#### 3.1 解法1（C++已过）

题目的要求是深拷贝（克隆），其实就是每个节点我们都要新建立节点而不是直接引用传进来的图中的节点。此解法是利用广度优先搜索，广度优先搜索是一层层搜索，我们因为要所有节点都新建立并且要将正确的邻居信息表示出来（正确的边的关系），所以可以借用哈希表存对应节点信息（题目节点对应克隆节点）。

本人代码：

|  |
| --- |
| /\*  // Definition for a Node.  class Node {  public:      int val;      vector<Node\*> neighbors;        Node() {          val = 0;          neighbors = vector<Node\*>();      }        Node(int \_val) {          val = \_val;          neighbors = vector<Node\*>();      }        Node(int \_val, vector<Node\*> \_neighbors) {          val = \_val;          neighbors = \_neighbors;      }  };  \*/  class Solution {  public:      //广度优先搜索克隆图      Node\* cloneGraph(Node\* node) {           if (node == nullptr) {              return node;          }          unordered\_map<Node\*,Node\*> s;          // 将题目给定的节点添加到队列          queue<Node\*> q;          q.push(node);          // 克隆第一个节点并存储到哈希表中          s[node] = new Node(node -> val);          // 广度优先搜索          while(!q.empty()) {              // 取出队列的头节点              Node\* t = q.front();              q.pop();              // 遍历该节点的邻居              for(int i = 0; i < t -> neighbors.size(); i++) {                  if(s.find(t -> neighbors[i]) == s.end()) {                      // 将邻居节点加入队列中                      q.push(t -> neighbors[i]);                       // 如果没有被访问过，就克隆并存储在哈希表中                      s[t -> neighbors[i]] = new Node(t -> neighbors[i] -> val);                  }                  // 更新当前节点的邻居列表                  s[t] -> neighbors.push\_back(s[t -> neighbors[i]]);              }          }          return s[node];      }  }; |

#### 3.2 解法2（C++已过）

此解法是利用深度优先搜索，深度优先搜索是不断向下搜索，我们因为要所有节点都新建立并且要将正确的邻居信息表示出来（正确的边的关系），所以可以借用哈希表存节点信息（节点的值不会重复，所以哈希表是节点值对应克隆节点）。

本人代码：

|  |
| --- |
| /\*  // Definition for a Node.  class Node {  public:      int val;      vector<Node\*> neighbors;        Node() {          val = 0;          neighbors = vector<Node\*>();      }        Node(int \_val) {          val = \_val;          neighbors = vector<Node\*>();      }        Node(int \_val, vector<Node\*> \_neighbors) {          val = \_val;          neighbors = \_neighbors;      }  };  \*/  class Solution {  public:      //深度优先搜索克隆图      unordered\_map<int,Node\*> s;      Node\* cloneGraph(Node\* node) {          if(node == nullptr) {              return nullptr;          }          // 克隆节点，注意到为了深拷贝我们不会克隆它的邻居的列表          Node\* cloneNode = new Node(node -> val);           // 哈希表存储          s[node -> val] = cloneNode;          // 遍历该节点的邻居并更新克隆节点的邻居列表          for(int i = 0; i < node -> neighbors.size(); i++) {              // 如果该节点没有被访问过，则继续深度优先搜索              if(s.find(node -> neighbors[i] -> val) == s.end())                  cloneNode -> neighbors.push\_back(cloneGraph(node -> neighbors[i]));              // 如果该节点已经被访问过，则直接从哈希表中取出对应的克隆节点              else                  cloneNode -> neighbors.push\_back(s[node -> neighbors[i] -> val]);          }          return cloneNode;      }  }; |

## [486. 预测赢家](https://leetcode-cn.com/problems/predict-the-winner/)

### 1 原题目

给定一个表示分数的非负整数数组。 玩家 1 从数组任意一端拿取一个分数，随后玩家 2 继续从剩余数组任意一端拿取分数，然后玩家 1 拿，…… 。每次一个玩家只能拿取一个分数，分数被拿取之后不再可取。直到没有剩余分数可取时游戏结束。最终获得分数总和最多的玩家获胜。

给定一个表示分数的数组，预测玩家1是否会成为赢家。你可以假设每个玩家的玩法都会使他的分数最大化。

**示例 1：**

输入：[1, 5, 2]

输出：False

解释：一开始，玩家1可以从1和2中进行选择。

如果他选择 2（或者 1 ），那么玩家 2 可以从 1（或者 2 ）和 5 中进行选择。如果玩家 2 选择了 5 ，那么玩家 1 则只剩下 1（或者 2 ）可选。

所以，玩家 1 的最终分数为 1 + 2 = 3，而玩家 2 为 5 。

因此，玩家 1 永远不会成为赢家，返回 False 。

**示例 2：**

输入：[1, 5, 233, 7]

输出：True

解释：玩家 1 一开始选择 1 。然后玩家 2 必须从 5 和 7 中进行选择。无论玩家 2 选择了哪个，玩家 1 都可以选择 233 。

最终，玩家 1（234 分）比玩家 2（12 分）获得更多的分数，所以返回 True，表示玩家 1 可以成为赢家。

**提示：**

1 <= 给定的数组长度 <= 20.

数组里所有分数都为非负数且不会大于 10000000 。

如果最终两个玩家的分数相等，那么玩家 1 仍为赢家。

### 2 基础知识

递归算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

本题可以采用递归的解法，我们设置玩家一拿到的分数为正数，玩家二拿到的分数设置为负数（用一个turn来设置分数的正负），这样就可以通过加法运算来判定玩家一是否可以获胜，如果最后二者分数加起来大于等于0，那么玩家一获胜，否则就是玩家二获胜。每个玩家都可以选择数组的头部或者尾部（最后选择的是对自己有利的选择）

对于递归函数最后返回的max(chooseHead \* turn,chooseTail \* turn) \* turn我做一下解释。假设数组为[1,5,2],假如果此时玩家一已经选择了头部的1，进入下层递归，玩家二会选择5，那么分数就是-5+2=-3，并返回到上一层。假如果此时玩家一已经选择了尾部的2，进入下层递归，玩家二会选择5，那么分数就是-5+1=-4，并返回到上一层。-3返回之后还要加上1（头部）就是-2，-4返回之后还要加上2（尾部）就是-2。二者都是-2，所以此时玩家一无论怎么选择都是输了。其实我们可以发现往下递归其实都是选择当前对自己有利的，对自己有利的就是向下递归和大的那种。两个数字比较就三种情况（全正，全负，一正一负），全正和一正一负都好比较，两个负数比较则是靠近0的那端更大，如果此时是玩家二选择（玩家二是负分）他要到达的效果是分数离0远的，所以我们比较的时候可以利用上turn这个变量，比较max(chooseHead \* turn,chooseTail \* turn)就是对的，最后在乘上turn是为了使分数符号变成原来的符号（因为max比较的时候改变了它的符号）。

本人代码：

|  |
| --- |
| class Solution {  public:      //递归解法      //玩家一的分数是正数，玩家二得到的分数是负数，turn的正负代表的是几号玩家      //将两个玩家选择的分数相加如果最后的结果是正数就代表玩家一胜利      int total(vector<int>& nums, int start, int end, int turn) {          if (start == end) {              return nums[end] \* turn;          }          //当前玩家选择头元素和尾元素的情况          int chooseHead = nums[start] \* turn + total(nums,start + 1,end,-turn);          int chooseTail = nums[end] \* turn + total(nums,start,end - 1,-turn);          //返回大的那一种情况          return max(chooseHead \* turn,chooseTail \* turn) \* turn;      }      bool PredictTheWinner(vector<int>& nums) {          return total(nums,0,nums.size() - 1,1) >= 0;      }  }; |

## [201. 数字范围按位与](https://leetcode-cn.com/problems/bitwise-and-of-numbers-range/)

### 1 原题目

给定范围 [m, n]，其中 0 <= m <= n <= 2147483647，返回此范围内所有数字的按位与（包含 m, n 两端点）。

**示例 1:**

输入: [5,7]

输出: 4

**示例 2:**

输入: [0,1]

输出: 0

### 2 基础知识

位运算（按位与）（解法一用到）

Brian Kernighan 算法（布赖恩·克尼根算法）（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

先来说一下按位与操作，两个数字的对应位置上的二进制数字都是1此位置才是1，否则就是0。

我来说说我的理解：Brian Kernighan 算法就是当n和n-1进行按位与运算时，会将n的二进制表示的最后一位1变成0。（就是个运算规律其实）。此题目是要求你将范围内[m,n]的所有数字进行按位与运算。对所有数字执行按位与运算的结果是所有对应二进制字符串的公共前缀再用零补上后面的剩余位（证明在链接里）。利用此算法就是可以快速找到公共前缀（每次都是将最右侧的1换成0，所以数字跳的也多），运算截至的条件就是n<=m

我的理解将这些，没太理解透，先这样吧。

参考链接：https://leetcode-cn.com/problems/bitwise-and-of-numbers-range/solution/shu-zi-fan-wei-an-wei-yu-by-leetcode-solution/

本人代码：

|  |
| --- |
| class Solution {  public:      int rangeBitwiseAnd(int m, int n) {          while (m < n) {              //抹去最右边的 1              n = n & (n - 1);          }          return n;      }  }; |

## [332. 重新安排行程](https://leetcode-cn.com/problems/reconstruct-itinerary/)

### 1 原题目

给定一个机票的字符串二维数组 [from, to]，子数组中的两个成员分别表示飞机出发和降落的机场地点，对该行程进行重新规划排序。所有这些机票都属于一个从 JFK（肯尼迪国际机场）出发的先生，所以该行程必须从 JFK 开始。

**提示：**

如果存在多种有效的行程，请你按字符自然排序返回最小的行程组合。例如，行程 ["JFK", "LGA"] 与 ["JFK", "LGB"] 相比就更小，排序更靠前

所有的机场都用三个大写字母表示（机场代码）。

假定所有机票至少存在一种合理的行程。

所有的机票必须都用一次 且 只能用一次。

**示例 1：**

输入：[["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

输出：["JFK", "MUC", "LHR", "SFO", "SJC"]

**示例 2：**

输入：[["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

输出：["JFK","ATL","JFK","SFO","ATL","SFO"]

解释：另一种有效的行程是 ["JFK","SFO","ATL","JFK","ATL","SFO"]。但是它自然排序更大更靠后。

### 2 基础知识

Hierholzer 算法,欧拉路径（解法一用到）

DFS深度优先搜索算法，回溯（解法二用到）

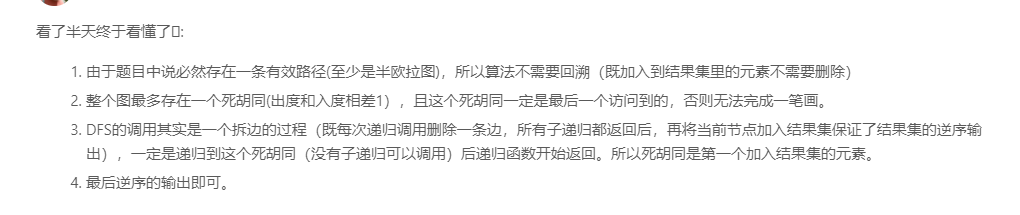
### 3 解法

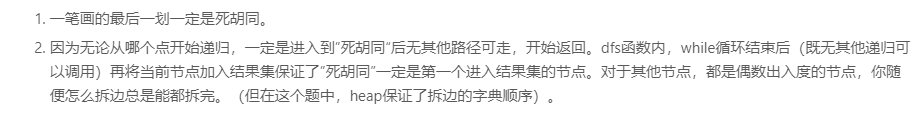
#### 3.1 解法1（C++已过）

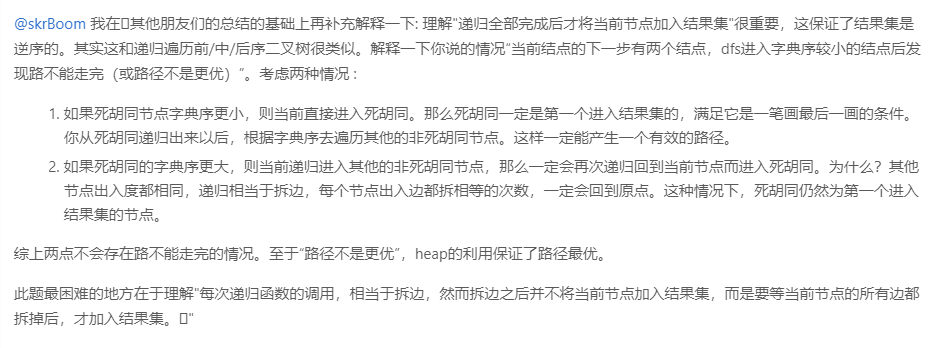
Hierholzer 算法很happy啊。

一笔画，就是所有路径从起点出发一笔走完，需要注意的是这里会有死胡同（进到这点就出不去了），死胡同本题最多有一个。死胡同一定是最后一个走到的，提前走到就会无法完成一笔画这个任务。为了让死胡同的那个点是最后经过的，我们可以规定下点加到结果集的要求：那就是当前节点的所有路径都已经遍历完了，才将当前节点入到结果中，这样就能保证死胡同那点第一个进入结果中（结果最后要翻转下的）。

对于当前节点而言，从它的每一个非死胡同分支出发进行深度优先搜索，都将会搜回到当前节点。而从它的死胡同分支出发进行深度优先搜索将不会搜回到当前节点。也就是说当前节点的死胡同分支将会优先于其他非死胡同分支入结果中。







参考链接：https://leetcode-cn.com/problems/reconstruct-itinerary/solution/zhong-xin-an-pai-xing-cheng-by-leetcode-solution/

本人代码：

|  |
| --- |
| class Solution {  public:      //记录表的关系，并利用优先队列排序      unordered\_map<string,priority\_queue<string, vector<string>, greater<string>>> m;      vector<string> res;      //深度优先搜索      void dfs(string curr) {          while (m[curr].size() > 0 && m.find(curr) != m.end()) {              string temp = m[curr].top();              m[curr].pop();              dfs(temp);          }          res.push\_back(curr);      }      vector<string> findItinerary(vector<vector<string>>& tickets) {          for (int i = 0; i < tickets.size(); i++) {              m[tickets[i][0]].push(tickets[i][1]);          }          dfs("JFK");          reverse(res.begin(), res.end());          return res;      }  }; |

#### 3.1 解法2（C++已过）

此解法就是利用深度优先搜索来找寻结果，利用优先队列容器可以方便的将到达站进行排序，然后就是正常回溯+DFS解题了，唯一一个需要注意的点就是我代码中的高亮部分，此部分限制了循环次数，如果不加上此限制可能会导致死循环导致程序跑不过（超时）

关于优先队列讲解的链接：<https://blog.csdn.net/weixin_36888577/article/details/79937886>

本人代码：

|  |
| --- |
| class Solution {  public:      int n;      unordered\_map<string,queue<string>> m;      vector<string> res;      bool dfs(string curr) {          if(res.size() == n + 1)              return true;          if(m[curr].size() == 0)              return false;          int size = m[curr].size();          int currsize = 0;          while (m[curr].size() > 0 && m.find(curr) != m.end() && currsize < size) {              currsize++;              string temp = m[curr].front();              m[curr].pop();              res.push\_back(temp);              if(dfs(temp))                  return true;              else {                  res.pop\_back();                  m[curr].push(temp);              }          }          return false;      }      vector<string> findItinerary(vector<vector<string>>& tickets) {          n = tickets.size();          unordered\_map<string,priority\_queue<string, vector<string>, greater<string>>> m1;          for (int i = 0; i < n; i++) {              m1[tickets[i][0]].push(tickets[i][1]);          }          for (auto q : m1) {              string s = q.first;              while (m1[s].size()) {                  m[s].push(m1[s].top());                  m1[s].pop();              }          }          res.push\_back("JFK");          dfs("JFK");          return res;      }  }; |

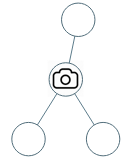
## [968. 监控二叉树](https://leetcode-cn.com/problems/binary-tree-cameras/)

### 1 原题目

给定一个二叉树，我们在树的节点上安装摄像头。

节点上的每个摄影头都可以监视其父对象、自身及其直接子对象。

计算监控树的所有节点所需的最小摄像头数量。

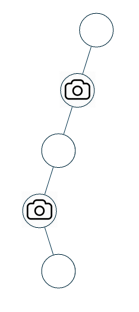


**示例 1：**

输入：[0,0,null,0,0]

输出：1

解释：如图所示，一台摄像头足以监控所有节点。



**示例 2：**

输入：[0,0,null,0,null,0,null,null,0]

输出：2

解释：需要至少两个摄像头来监视树的所有节点。 上图显示了摄像头放置的有效位置之一。

**提示：**

给定树的节点数的范围是 [1, 1000]。

每个节点的值都是 0。

### 2 基础知识

贪心算法，递归（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题的贪心就是安装的监控要在叶子结点的父结点上，这样安装监控才会使监控所能监控的节点数目最大化，并且需要从树的下面往上推导。推导出节点的状态来计算监控数量。

参考链接：https://leetcode-cn.com/problems/binary-tree-cameras/solution/968-jian-kong-er-cha-shu-di-gui-shang-de-zhuang-ta/

本人代码：

|  |
| --- |
| /\*\*   \* Definition for a binary tree node.   \* struct TreeNode {   \*     int val;   \*     TreeNode \*left;   \*     TreeNode \*right;   \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}   \* };   \*/  class Solution {  public:      int res;      int postorder(TreeNode\* root) {          // 空节点，该节点有覆盖          if (root == NULL)              return 2;          int left = postorder(root -> left);          int right = postorder(root -> right);          // 情况1          // 左右节点都有覆盖          if (left == 2 && right == 2)              return 0;           // 情况2          // left == 0 && right == 0 左右节点无覆盖          // left == 1 && right == 0 左节点有摄像头，右节点无覆盖          // left == 0 && right == 1 左节点有无覆盖，右节点摄像头          // left == 0 && right == 2 左节点无覆盖，右节点覆盖          // left == 2 && right == 0 左节点覆盖，右节点无覆盖          else if (left == 0 || right == 0) {              res++;              return 1;          }          // 情况3          // left == 1 && right == 2 左节点有摄像头，右节点有覆盖          // left == 2 && right == 1 左节点有覆盖，右节点有摄像头          // left == 1 && right == 1 左右节点都有摄像头          // 其他情况前段代码均已覆盖          else if (left == 1 || right == 1)              return 2;          return -1;      }      int minCameraCover(TreeNode\* root) {          res = 0;          // 情况4,root无覆盖          if (postorder(root) == 0)              res++;          return res;      }  }; |

## [546. 移除盒子](https://leetcode-cn.com/problems/remove-boxes/)

### 1 原题目

给出一些不同颜色的盒子，盒子的颜色由数字表示，即不同的数字表示不同的颜色。

你将经过若干轮操作去去掉盒子，直到所有的盒子都去掉为止。每一轮你可以移除具有相同颜色的连续 k 个盒子（k >= 1），这样一轮之后你将得到 k\*k 个积分。

当你将所有盒子都去掉之后，求你能获得的最大积分和。

**示例：**

输入：boxes = [1,3,2,2,2,3,4,3,1]

输出：23

解释：

[1, 3, 2, 2, 2, 3, 4, 3, 1]

----> [1, 3, 3, 4, 3, 1] (3\*3=9 分)

----> [1, 3, 3, 3, 1] (1\*1=1 分)

----> [1, 1] (3\*3=9 分)

----> [] (2\*2=4 分)

**提示：**

1 <= boxes.length <= 100

1 <= boxes[i] <= 100

### 2 基础知识

动态规划（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

大家自己去看官方题解吧：<https://leetcode-cn.com/problems/remove-boxes/solution/yi-chu-he-zi-by-leetcode-solution/>

我是讲不清，看完吧，明白一句话“道理我都懂”，哈哈哈哈，估计这题目，几天后忘得一干二净。

本人代码：

|  |
| --- |
| class Solution {  public:      int dp[100][100][100];      int removeBoxes(vector<int>& boxes) {          memset(dp, 0, sizeof(dp));          return calculatePoints(boxes, 0, boxes.size() - 1, 0);      }      int calculatePoints(vector<int>& boxes, int l, int r, int k) {          if (l > r)              return 0;          if (dp[l][r][k] != 0)              return dp[l][r][k];          dp[l][r][k] = calculatePoints(boxes, l, r - 1, 0) + (k + 1) \* (k + 1);          for (int i = l; i < r; i++) {              if (boxes[i] == boxes[r]) {                  dp[l][r][k] = max(dp[l][r][k], calculatePoints(boxes, l, i, k + 1) + calculatePoints(boxes, i + 1, r - 1, 0));              }          }          return dp[l][r][k];      }  }; |

## 题

### 1 原题目

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

本人代码：

|  |
| --- |
|  |

## 题

### 1 原题目

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

本人代码：

|  |
| --- |
|  |