1. 两数之和

给定一个整数数组 `nums` 和一个目标值 `target`，请你在该数组中找出和为目标值的那 **两个** 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

示例:

```
给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]
```

经典的哈希，你只需要记录每个数字的任意一个出现位置即可。

每次检查target-nums[i]是否存在，如果存在那么它的哈希值是否等于i。

3. 无重复字符的最长子串

给定一个字符串，请你找出其中不含有重复字符的 **最长子串** 的长度。

示例 1:

```
输入: "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

示例 2:

```
输入: "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
```

示例 3:

```
输入: "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
```

双指针，每次固定左端点，在符合条件的情况下，让右端点能走多远走多远。

左右端点都定下来之后更新答案，保存字符是否出现过可以利用256位的哈希。

4. 寻找两个正序数组的中位数

给定两个大小为 m 和 n 的正序（从小到大）数组 `nums1` 和 `nums2`。

请你找出这两个正序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。

你可以假设 `nums1` 和 `nums2` 不会同时为空。

示例 1:

```
nums1 = [1, 3]
nums2 = [2]

则中位数是 2.0
```

示例 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

则中位数是 (2 + 3)/2 = 2.5
```

究极dfs

首先假设题目给了我们两个数组 A 和 B ，他们的长度分别为n和m，简要示意如下：

```
A：a[1] a[ 2 ] ... a[ n ]
B：b[1] b[ 2 ] ... b[ m ]
```

一种比较朴素的想法是：由于这两个数组都是有序的，我们可以采用双指针合并的方式在 O(n+m) 的时间内把它们合并成一个新的有序数组 C ，简要示意如下：

```
C：c[ 1 ] c[ 2 ] ... c[ n + m ]
```

此时又分成两种情况：

1. n+m是偶数。此时中位数是：
   ```
   ( c[ ( n + m ) / 2 ] + c[ ( n + m ) / 2 + 1 ] ) / 2
   ```
2. n+m是奇数。此时中位数是：
   ```
   c[ ( n + m ) / 2 + 1 ]
   ```

但是到这儿会发现实际上并不需要合并这两个数组，我们只需要弄一个算法在 O(log(n+m)) 的时间内在这两个数组中找到排第k位的数字即可。

那么怎么构造这种算法呢?

如果我们比较a[k / 2]与b[k / 2]的大小，有以下三种情况：

1. a[k / 2]<b[k / 2]。此时考虑a[k / 2]，即使[b[1] , b[k / 2 - 1]]这个区间内的所有数字都小于a[k / 2]，那么在整个合并后的数组 C 中也只有(k / 2 - 1) * 2 = k - 2个数字小于a[k / 2]，这也就说明a[k / 2]的排名最高为第 k - 1 名，它绝对不可能是我们要寻找的第 k 位数字。由于两个数组都是有序的，它前面的数字更不可能是我们要寻找的答案。
2. a[k / 2]==b[k / 2]。二者都有可能是第 k 位数字。
3. a[k / 2]>b[k / 2]。情况与1相反，b[k / 2]绝对不可能是第 k 位数字。

那么我们是不是可以考虑在每一次判断中删去 k / 2 个不可能的数字（例如上述情况1中的 [a [0] , a [k / 2]]）呢？这样的话在 O(log(k)) 的时间内我们就能找到数组 C 中第 k 位的数字。又因为 k = (n + m) / 2，总体的时间复杂度就变成了 O(log(n+m)) ！

实际写代码时主要有以下几个考虑：

- 数字的删去可以控制左右指针来完成，每次只考虑指针区间内的数字即可。
- 当比较的两个数字相等时，删去哪一部分都无所谓，不会影响最后结果。
- 当 k = 1 时，意味着只要返回当前比较的两个数字中的较小值即可，不需要再进行 k / 2 的计算。
- 有时候会遇到某个数组已经被全部删除的情况，这时我们直接返回剩下那个数组中的第 k 位数字即可。参考代码如下：
  ```
  if(l1>r1) return (double)b[l2+k-1];
  if(l2>r2) return (double)a[l1+k-1];
  ```
- 如果当前有某个数组的长度小于 k / 2 ，那么我们只需要比较这个数组的最后一位即可。
- 注意k每次并不是严格减少 k / 2 的，它减少的量取决于数组中被删除的数字个数。删除a数组时参考代码如下：
  ```
  L=min(r1,l1+k/2-1);//L是a数组新的边界
  dfs(L+1,r1,l2,r2,k-(L-l1+1));
  ```
- 为了排除 n + m 奇偶性的影响，可以寻找排名为 (n + m + 1) / 2 和 (n + m + 2) / 2的两个数字。当 n + m 为奇数时，上述两个式子会获得相同的结果；当 n + m为偶数时，上述两个式子对应的两个数字就是中位数左右两边的数字。

---

5. 最长回文子串

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”
```

经典马拉车，O(n)时间内解决最长回文子串。

为了处理奇偶回文串的问题，马拉车要把原串用#完全分隔开，像下面这样：

- abba->#a#b#b#a#
- aba->#a#b#a#

这里注意我在写代码的时候为了让下标从1开始，还在处理过后的字符串开始加了一个$作为占位符。

然后需要保存三样东西，c是当前延伸到最靠右的回文串的回文中心，r是当前延伸到最靠右的回文串的终点位置，p[i]是i点的回文半径——即i左边与右边有多少长度的字符串是相等的。

这里注意我在写代码的时候让p[i]最小值为1，即无论如何该字符本身都能构成一个半径为1的回文串。

接下来考虑怎样利用c来求得p[i]。由于c的回文特性，不难发现p[i]与p[j]应该是相等的，这里的j是i关于c的对称点，j=2*c-i。

但是有一种特殊的情况，如果p[j]比r-i还要大，即p[j]推断的范围超出了c能确定的范围，我们就只能确定p[i]至少等于r-i。

那么可以写出下面的代码：

```
p[i]=min(r-i,p[2*c-i]);
```

完成了推断之后，还要对p[i]进一步地扩展以确保答案的正确性。代码如下：

```
while(ss[i-p[i]]==ss[i+p[i]]) p[i]++;
```

最后别忘记更新回文中心c和回文终点r。代码如下：

```
if(i+p[i]>r)
    c=i,r=i+p[i];
```

最后返回答案的时候还有一点需要注意的，在找到p[pos]=mx使得mx是p[i]中的最大值之后，对应的原字符串的子串为：

```
substr((pos-mx)/2,mx-1)
```

此时不需要对原字符串进行任何改动。

---

9. 回文数

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?

水题。

回文数有一种不把数字转化为字符串的解法。你可以这样来获取 x 的后半部分：

```
        while(x>y) {
            y*=10;
            y+=x%10;
            x/=10;
        }
```

如果x长度为奇数，那么我们只需要考虑 y / 10 跟 x 是否相等即可，所以返回的答案是这样：

```
        return x==y||x==y/10;
```

还有一些特判：负数、个位数和末尾是 0 的数字：

```
        if(x<0) return 0;
        if(x/10==0) return 1;
        if(x%10==0) return 0;
```

---

10. 正则表达式匹配

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
```

大DP，很有价值。

我们考虑做一个dp[i][j]，表示s匹配到了i位置并且p匹配到了j位置。

分两种情况考虑：

- p[j]是星号： 此时需要考虑s[i]与p[j-1]是否匹配。

  如果匹配，那么有两种选择方案：一种是把s[i]交给p[j-1]来匹配，那么dp[i][j]=dp[i-1][j]；另一种是不把s[i]交给p[j-1]匹配，那么dp[i][j]=dp[i][j-2]

  如果s[i]与p[j-1]不匹配，那么只能把s[i]交给p[j-2]匹配，那么dp[i][j]=dp[i][j-2]

- p[j]不是星号： 此时直接检查s[i]与p[j]是否匹配即可，方程：

  dp[i][j]=dp[i-1][j-1]（s[i]与p[j]匹配） dp[i][j]=0（s[i]与p[j]不匹配）

有两个注意点：

1. 初值的设定：如果按照上面那样设定dp[i][j]中的i和j为当前匹配位置的话，那么就需要考虑一个问题：对于p串开头的那些形如“a*”的串，它们都可以被匹配为空串。这意味着dp[0][j]需要进行处理，对这些空串位置都是可以匹配的。
2. p串的处理：如果连续出现了多个*，应当对它们去重。但是本题给定的都是合法串，所以不需要去重。

---

14. 最长公共前缀

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 `""`。

示例 1:

```
输入：["flower","flow","flight"]
输出："fl"
```

示例 2:

```
输入：["dog","racecar","car"]
输出：""
解释：输入不存在公共前缀。
```

说明:

所有输入只包含小写字母 `a-z` 。

水题。

维护一个当前最大前缀，每一次用当前的前缀跟现在的字符串匹配就行。复杂度O(总长度)。

---

15. 三数之和

给你一个包含 *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 ？ 请你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。

示例：

```
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]
]
```

双指针

首先做一个排序，对每个数字向后二分左端点 l 和右端点 r 。

分三种情况：

1. 如果等于 0： 更新答案，l 和 r 全部向内缩进。
2. 如果大于 0： r 向内缩进，把答案变小。
3. 如果小于 0： l 向内缩进，把答案变大。

注意处理相同数字的情况，总复杂度在 O(n^2) 附近。

---

31. 下一个排列

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 (https://baike.baidu.com/item/%E5%8E%9F%E5%9C%B0%E7%AE%97%E6%B3%95)**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`

找下一个排列的基本思想是找到最靠右的正序对a[i-1]<a[i]。

然后找到最靠右的a[j]>a[i-1]，交换a[i-1]和a[j]。

以上操作相当于把当前排列变得更大。

为了保证交换后当前排列最小，还需要把[i,n-1]区间内的数字进行排序。

别忘了特判找不到正序对的情况，此时直接返回原数组的reverse即可。

---

41. 缺失的第一个正数

给你一个未排序的整数数组，请你找出其中没有出现的最小的正整数。

示例 1:

```
输入: [1,2,0]
输出: 3
```

示例 2:

```
输入: [3,4,-1,1]
输出: 2
```

示例 3:

```
输入: [7,8,9,11,12]
输出: 1
```

提示:

你的算法的时间复杂度应为O(*n*)，并且只能使用常数级别的额外空间。

思维题。

主要思路还是用某个位置上的正负号来表示这个位置上的数字是否出现过。

因为一个n大小的数组中最小的未出现整数，答案一定在[1,n+1]之间，所以用给定的数组是可以实现原地哈希的。

细节比较多，罗列如下：

1. 负数应当直接处理成n+1，这样不会影响答案。因为哈希时我们只考虑[1,n]区间内的数字。
2. 在数组中所有数字已经保证为正数的情况下开始算法，流程中如果遇见了负数就不要再把它乘以负一了，因为此时代表该位置上的数字已经出现过。
3. 流程中记得取绝对值防止下标越界。
4. 第一个为正数的位置就是答案。如果发现[1,n]以内的数字全部出现过，返回n+1。

---

54. 螺旋矩阵

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]
```

联动：https://leetcode-cn.com/problems/shun-shi-zhen-da-yin-ju-zhen-lcof/submissions/ (https://leetcode-cn.com/problems/shun-shi-zhen-da-yin-ju-zhen-lcof/submissions/)

---

59. 螺旋矩阵 II

给定一个正整数 *n*，生成一个包含 1 到 n^2 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 ]
]
```

联动：https://leetcode-cn.com/problems/shun-shi-zhen-da-yin-ju-zhen-lcof/submissions/ (https://leetcode-cn.com/problems/shun-shi-zhen-da-yin-ju-zhen-lcof/submissions/)

本题是顺时针填数，联动题是顺时针取数。

---

67. 二进制求和

给你两个二进制字符串，返回它们的和（用二进制表示）。

输入为 **非空** 字符串且只包含数字 `1` 和 `0`。

示例 1:

```
输入: a = "11", b = "1"
输出: "100"
```

示例 2:

```
输入: a = "1010", b = "1011"
输出: "10101"
```

提示:

- 每个字符串仅由字符 `'0'` 或 `'1'` 组成。
- `1 <= a.length, b.length <= 10^4`
- 字符串如果不是 `"0"` ，就都不含前导零。

水题。

把a和b都倒过来，补位成长度一样，由小到大累加即可。

可以原地修改，注意特判最后一位进位的情况。

注意返回之前要把a倒回来。

---

70. 爬楼梯

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
```

斐波那契

节约空间的话可以用两个变量，节约时间的话可以**矩阵快速幂。**

76. 最小覆盖子串

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。

滑动窗口，维护当前窗口里有哪些字符，一旦符合条件马上尝试更新答案，然后让左端点向右移动一格。

由于右端点不会倒退，总体时间复杂度是O(2*n)+O(check)。

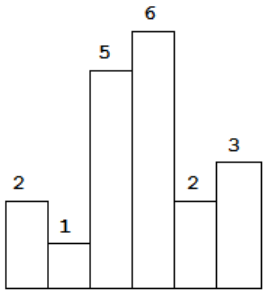
有三个注意点：

1. 本题中s子串包含t所有字符的含义是s子串中每个字符的出现次数必须大于等于t中每个字符的出现次数，所以s="a"，t="aa"时应该无解。
2. char的范围是[-128,127]，但是可见字符范围只从[32,126]，在map中查找不可见字符可能会出很多问题。
3. map太慢了，对于char这种小范围数据应该使用数组储存哈希值来实现O(1)查询。

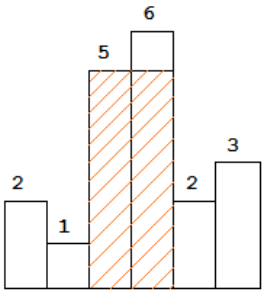
84. 柱状图中最大的矩形

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
```

单调栈

对每个孤立的点 i 进行考虑，它对答案的贡献实际上是 height[i] * (r[i] - l [i] - 1) 。这里的 l[i] 是 i 左边第一个小于 height[i] 的位置，r[i] 则是右边第一个小于的。

考虑如何找到这样的区间，维护一个 height 单调递增的 stack ， stack里面存储下标。遍历到 i 下标时，有以下两种情况：

1. 当某个数字 x 被 pop 时：这个数字已经找到了 l[x] = 栈顶的下一位 以及 r[x] = i ，直接更新答案。
2. 当遍历结束后某个数字 y 仍在 stack 中时：l[y] 同上，但是 r[y] = n ，意味着 y 右边没有比 y 更小的数字。

有这样一个事实：被 pop 的数字一定比它左边第一个未被 pop 的数字的 height 要大，也比它右边第一个未被 pop 的数字的 height 要大。

为了方便处理下标，一开始在 stack 中加入 -1 。

85. 最大矩形

给定一个仅包含 0 和 1 的二维二进制矩阵，找出只包含 1 的最大矩形，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：6
```

联动：https://leetcode-cn.com/classic/problems/largest-rectangle-in-histogram/description/ (https://leetcode-cn.com/classic/problems/largest-rectangle-in-histogram/description/)

有了柱状图中最大矩形的 O(n) 做法，我们只需要维护每一行的信息然后对每一行做柱状图运算就完事了，总复杂度 O(n^2) 。

101. 对称二叉树

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
```

进阶:

你可以运用递归和迭代两种方法解决这个问题吗?

dfs

传两个参数下去，分别是 p 和 q 指针，两个指针向不同的方向走，判断一下 NULL 和值是否相等的情况即可。

105. 从前序与中序遍历序列构造二叉树

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
```

先序遍历根左右，中序遍历左根右。

根据先序遍历可以找到根，然后在中序遍历中找到根的左子树和右子树。

dfs时分别记录先序遍历的下标范围和中序遍历的下标范围，中序遍历的下标范围会被一分为二，然后根据左右子树的大小可以确定先序遍历的下标。

注意给定空的遍历时要返回NULL。

108. 将有序数组转换为二叉搜索树

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
```

dfs建树。

可以用dfs维护住左右区间，每次取出区间的中点作为根节点。

在保证序列有序的情况下，这样可以保证建出来的树是平衡二叉树。

124. 二叉树中的最大路径和

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
```

简单树形DP

方程：dp[i]=max{val[i],dp[L]+val[i],dp[R]+val[i]}

更新答案：ans=max(dp[i],dp[L]+dp[R]+val[i])

O(n)时间内就可以把上面两个全部处理出来了。

125. 验证回文串

给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。

示例 1:

```
输入："A man, a plan, a canal: Panama"
输出：true
```

示例 2:

```
输入: "race a car"
输出: false
```

水题。

把字母个数字抠出来判断回文，空字符串返回true。

126. 单词接龙 II

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：

1. 每次转换只能改变一个字母。
2. 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。
```

O(n^2) 建图， dij 扣出最短路构成的树， dfs 输出树。

注意可能会 T ， 玄学地把比较两个字符串的函数的形参改成**引用**就过了。

127. 单词接龙

给定两个单词（*beginWord* 和 *endWord*）和一个字典，找到从 *beginWord* 到 *endWord* 的最短转换序列的长度。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间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 0。
- 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: 5

解释: 一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog",
     返回它的长度 5。
```

示例 2:

```
输入:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

输出: 0

解释: endWord "cog" 不在字典中，所以无法进行转换。
```

联动：https://leetcode-cn.com/problems/word-ladder-ii/ (https://leetcode-cn.com/problems/word-ladder-ii/)

本题是简单版，只要求输出最短距离。

---

128. 最长连续序列

给定一个未排序的整数数组，找出最长连续序列的长度。

要求算法的时间复杂度为 *O(n)*。

示例:

```
输入: [100, 4, 200, 1, 3, 2]
输出: 4
解释: 最长连续序列是 [1, 2, 3, 4]。它的长度为 4。
```

并查集

要找到最长的连续序列，不妨遍历整个数组，如果数字 x 第一次出现，就把 x 和 x + 1 以及 x - 1 合并，合并时判断一下 x + 1 和 x - 1 的存在性。

注意这道题值域很大，只能用 unordered_map 来做并查集的父亲数组，判断键 x 是否存在时也不能判断 f[x] == 0 ，而要用 f.count(x) == 0 来判断。

---

139. 单词拆分

给定一个**非空**字符串 *s* 和一个包含**非空**单词列表的字典 *wordDict*，判定 *s* 是否可以被空格拆分为一个或多个在字典中出现的单词。

说明:

- 拆分时可以重复使用字典中的单词。
- 你可以假设字典中没有重复的单词。

示例 1:

```
输入: s = "leetcode", wordDict = ["leet", "code"]
输出: true
解释: 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。
```

示例 2:

```
输入: s = "applepenapple", wordDict = ["apple", "pen"]
输出: true
解释: 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。
     注意你可以重复使用字典中的单词。
```

示例 3：

```
输入: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
输出: false
```

---

trie和dp的结合。

一开始写了个爆搜，遇到以下数据就爆了：

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab

a aa aaa aaaa aaaaa

所以得换一种考虑方式，dp[i]表示前i位的字符串能否表示出来，那么可以得到下面的方程：

dp[i]=dp[i]|(dp[j]&&hav(s[i,j]))

hav用来检查给定字符串中是否存在对应单词，可以用trie来维护。

时间复杂度O(n^2)。

---

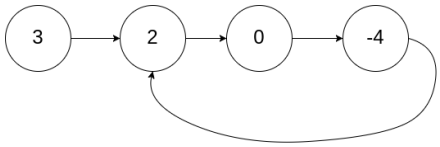
141. 环形链表

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。

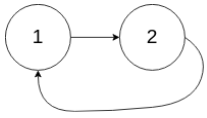
示例 1：

```
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？

快慢指针

用于 O(n) 在链表中找环。基本思想是：放一个指针 p ， 每次移动一位；再放一个指针 q ， 每次移动两位。如果 p 和 q 在某个位置相遇了，那么链表有环，否则无环。

如果链表中有环，那么哪怕是 n 长的环，由于 p 和 q 指针间速度差为 1 ， n 步之后二者一定能够相遇，时间复杂度就得到了保证。

146. LRU缓存机制

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 (https://baike.baidu.com/item/LRU)。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)` - 如果关键字 (key) 存在于缓存中，则获取关键字的值（总是正数），否则返回 -1。
写入数据 `put(key, value)` - 如果关键字已经存在，则变更其数据值；如果关键字不存在，则插入该组「关键字/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);    // 该操作会使得关键字 2 作废
cache.get(2);       // 返回 -1 (未找到)
cache.put(4, 4);    // 该操作会使得关键字 1 作废
cache.get(1);       // 返回 -1 (未找到)
cache.get(3);       // 返回  3
cache.get(4);       // 返回  4
```

双向链表

基本思路是用数组模拟链表， val[] 储存当前项的值， pre[] 储存上一项的指针， nxt[] 储存下一项的指针， fst 和 lst 标志链表起始和末尾， tot 和 C 分别标志当前容量和最大容量。

一些函数的说明如下：

1. mov(int key) 用于在执行操作后把某个节点放到链表头。
2. init(int key , int v) 插入第一个节点。
3. del() 删除尾节点。
4. add(int key , int v) 在头部插入一个新节点。

函数代码及注意事项如下：

1. fst 节点不需要再操作， lst 节点 mov() 之后要更新lst，其他都是对pre和nxt的一些双向链表基本操作。

```
void mov(int id) {
    if(id==fst) return;
    nxt[pre[id]]=nxt[id];
    if(id==lst) lst=pre[id];
    if(nxt[id]) pre[nxt[id]]=pre[id];
    pre[id]=0;
    nxt[id]=fst;
    pre[fst]=id;
    fst=id;
}
```

2. 额外考虑插入第一个节点的情况只要是为了考虑fst和lst的初始化，只需要更新 val[] 和修改 tot ， 无需对 pre[] 和 nxt[] 进行任何操作。

```
void init(int key,int v) {
     val[key]=v;
     fst=lst=key;
     ++tot;
 }
```

3. 记得更新 lst 和清空 val[] 免得后面的判断出错，还得把当前节点数 tot 减少一个。

```
void del() {
     int tmp=pre[lst];
     nxt[pre[lst]]=0;
     val[lst]=pre[lst]=nxt[lst]=0;
     lst=tmp;
     tot--;
 }
```

4. 注意头插节点之后移动 fst 指针和修改当前容量 tot 。

```
void add(int id,int v) {
     val[id]=v;
     pre[id]=0;
     nxt[id]=fst;
     pre[fst]=id;
     fst=id;
     ++tot;
 }
```

---

198. 打家劫舍

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 **不触动警报装置的情况下** ，一夜之内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 。
```

提示：

- `0 <= nums.length <= 100`
- `0 <= nums[i] <= 400`

树状数组维护dp，方程dp[i] = MAX{ dp[j] } (1 <= j <= i - 2)

注意树状数组是向上更新、向下求答案。

并且树状数组只能维护下标从 1 开始，否则会越界。

树状数组只能维护从 1 开始的区间的最大值，因为最大值不具有可加性。

维护区间最大值可以考虑线段树或者 ST 表。

注意判断输入数组是否为空。

206. 反转链表

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?

dfs递归向下，到达链表尾则向上回溯。

基本代码:

```
ListNode* dfs(ListNode* now) {
        if(now==NULL) return NULL;
        if(now->next==NULL) {
            lst=now;
            return now;
        }
        ListNode* pre=dfs(now->next);
        pre->next=now;
        now->next=NULL;
        return now;
    }
```

这个 lst 就是新链表的头节点，也是原链表的尾节点。

209. 长度最小的子数组

给定一个含有 **n** 个正整数的数组和一个正整数 **s ，** 找出该数组中满足其和 **≥ s** 的长度最小的连续子数组，并返回其长度**。**如果不存在符合条件的连续子数组，返回 0。

示例:

```
输入：s = 7, nums = [2,3,1,2,4,3]
输出：2
解释：子数组 [4,3] 是该条件下的长度最小的连续子数组。
```

进阶:

- 如果你已经完成了 *O*(*n*) 时间复杂度的解法, 请尝试 *O*(*n* log *n*) 时间复杂度的解法。

双指针。

固定左指针，移动右指针，再移动左指针。符合条件就更新答案。

215. 数组中的第K个最大元素

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。

水题，排序找一下就完事了。

---

221. 最大正方形

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
```

DP。方程是dp[i][j]=mmin(dp[i - 1][j],dp[i][j - 1],dp[i - 1][j - 1])+1;

dp[i][j]表示以 (i , j) 为右下角的位置最大的正方形边长，上述方程当且仅当matrix[i][j] 等于 1 的时候才使用。

最后扫一遍更新最大面积即可。

---

222. 完全二叉树的节点个数

给出一个**完全二叉树**，求出该树的节点个数。

说明:

完全二叉树 (https://baike.baidu.com/item/%E5%AE%8C%E5%85%A8%E4%BA%8C%E5%8F%89%E6%A0%91/7773232?fr=aladdin)的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2^h 个节点。

示例:

```
输入:
    1
   / \
  2   3
 / \  /
4  5 6

输出: 6
```

二分。

第一步找出树的深度，确定最底层，同时确定二分的上下界：

```
    int l=(1<<(dep-1)),r=(1<<dep)-1;
```

第二步二分最底层的最大序号，并且O(log(n))判断某个序号是否存在。

判断时候可以规定当前最底层的左边界和右边界，构成一个序列。如果目标处于序列前半段，向左走，否则向右走。如果左边界等于右边界代表找到了这个值。参考代码：

```
bool ok(TreeNode* now,int id,int L,int R) {
    if(L==R) return 1;
    int M=(L+R)/2;
    if(id<=M) {
        if(now->left==NULL) return 0;
        return ok(now->left,id,L,M);
    } else {
        if(now->right==NULL) return 0;
        return ok(now->right,id,M+1,R);
    }
}
```

总时间复杂度O(log(n)^2)。

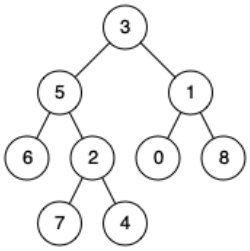
---

236. 二叉树的最近公共祖先

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 (https://baike.baidu.com/item/%E6%9C%80%E8%BF%91%E5%85%AC%E5%85%B1%E7%A5%96%E5%85%88/8918834?fr=aladdin)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。

倍增

基本原理是把一个十进制数转化为二进制存储，如10=8+2=1010。

原本把10减小到0需要十次，现在只需四次即可。

为此需要设定函数lg(x)，使得2的lg(x)次方大于等于x，便于二进制分解。

对一棵树找lca的朴素做法是两者同时向上找父亲，直到相等为止（首先需要令两者深度相等）。同一棵树中某个节点的父亲节点是唯一的，这样做一定可以使得两者在某一时刻相等，相等时即是找到了lca。

而倍增希望把向上找的一层一层上跳变成一次上跳2的n次方。

先进行一些预处理：

```
void dfs(int x,int f)
{
    dep[x]=dep[f]+1;
    fa[x][0]=f;
    for(int i=1;(1<<i)<=dep[x];++i)
        fa[x][i]=fa[fa[x][i-1]][i-1];
    for(auto y:e[x])
    {
        if(y==f) continue;
        dfs(y,x);
    }
}
```

其中dep是节点深度，fa[x][i]是x节点的第2的i次方（即是(1<<i)）辈祖先的编号。

显而易见的是x的2的i-1次方辈祖先的i-1次方辈祖先是x的2的(i-1)次方+2的(i-1)次方辈祖先，由此可以得出上述方程。

实际上与st表的原理十分类似。

注意：x的祖先数不可能超过它的深度。

随后是lca的设计：

```
int lg(int x){return (int)log2(x)+1;}
int lca(int x,int y)
{
    if(dep[x]<dep[y]) swap(x,y);
    if(dep[x]!=dep[y]){
        int dif=dep[x]-dep[y];
        for(int i=0;i<lg(dif);++i)
            if(dif&(1<<i))
                x=fa[x][i];
    }
    if(x==y)
        return x;
    for(int i=lg(dep[x])-1;i>=0;--i){
        if(fa[x][i]!=fa[y][i]){
            x=fa[x][i];
            y=fa[y][i];
        }
    }
    return fa[x][0];
}
```

---

238. 除自身以外数组的乘积

给你一个长度为 *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]
```

提示：题目数据保证数组之中任意元素的全部前缀元素和后缀（甚至是整个数组）的乘积都在 32 位整数范围内。

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。）

前后缀和

要求不使用除法求数组中除这个数字以外其他数字的乘积。

一个简单的想法是考虑前后缀积，答案就是： pre[i - 1] * suf[i + 1]

但是又要求了 O(1) 空间复杂度，不难发现其实前后缀积是可以通过递推得到的，写出来代码如下：

```
        for(int i=nums.size()-1;i>=1;--i)
            suf*=nums[i];
        for(int i=0;i<nums.size();++i) {
            ret.push_back(suf*pre);
            pre*=nums[i];
            if(i!=nums.size()-1) suf/=nums[i+1];//0
            else suf=1;
        }
```

但是可没这么简单，因为有 0 的存在，导致上面的除法可能会出现除 0 错误。不妨分析一下 0 的出现情况：

1. 没有 0 ： 那么上面的算法是完全正确的。
2. 有一个 0 ： 只有 0 那个位置的答案是前后缀积，其他地方都是 0 。
3. 有两个及以上 0 ： 全是 0 。

特判一下即可。

---

239. 滑动窗口最大值

给定一个数组 *nums*，有一个大小为 *k* 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 *k* 个数字。滑动窗口每次只向右移动一位。

返回滑动窗口中的最大值。

进阶：

你能在线性时间复杂度内解决此题吗?

示例:

```
输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3
输出: [3,3,5,5,6,7]
解释:

  滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

提示：

- `1 <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`
- `1 <= k <= nums.length`

---

deque

做一个递减的双向队列，最大值放在队头，新的值从队尾加。

队列里存的都是下标，注意要把窗口外的最大值给 pop 了。

---

287. 寻找重复数

给定一个包含 *n* + 1 个整数的数组 *nums*，其数字都在 1 到 *n* 之间（包括 1 和 *n*），可知至少存在一个重复的整数。假设只有一个重复的整数，找出这个重复的数。

示例 1:

```
输入: [1,3,4,2,2]
输出: 2
```

示例 2:

```
输入: [3,1,3,4,2]
输出: 3
```

说明:

1. **不能**更改原数组（假设数组是只读的）。
2. 只能使用额外的 $O(1)$ 的空间。
3. 时间复杂度小于 $O(n^2)$ 。
4. 数组中只有一个重复的数字，但它可能不止重复出现一次。

思路比较多，主要有三个：

1. **二分。复杂度 O(nlog(n))**

假设 cnt(i) 等于数组 nums 中小于等于i的数字个数，那么只需要找到最小的 i 使得 cnt(i) > i 即可，显然 cnt() 是一个单调函数。 如果重复数字超过三个，那么可以考虑成重复数字替换掉了另外的数字。对于那些小于它的数字， cnt() 可能会变小或者不变，不影响二分结果；对于那些大于它的数字，cnt() 还是大于重复数字的 cnt()，也不会对答案造成影响。

```
    int l=1,r=n;
    while(l<=r) {
        int m=(l+r)/2;
        int cnt=0;
        for(auto i:nums) cnt+=(i<=m);
        if(cnt>m) r=m-1;
        else l=m+1;
    }
    return l;
```

2. **二进制分解。复杂度 O(nlog(n))**

这种做法充分利用了值域的特性。统计两个数字 x[i] 和 y[i]： x[i] 代表 [1 , n] 中有多少个数字在第 i 位上是 1 ， y[i] 则代表数组 nums 中有多少个数字在第 i 位上是 1 。 不难发现只有当 x[i] > y[i] 时重复数字在该位上才是 1 ，因为重复的数字必定会导致这一位上 1 的个数多一个。这样逐位操作就可以还原出重复的数字。

```
int ans=0;
for(int i=0;i<20;++i) {
    int x=0,y=0;
    for(int j=1;j<=n;++j)
        if(j&(1<<i)) ++y;
    for(auto j:nums)
        if(j&(1<<i)) ++x;
    if(x>y&&y!=0) ans+=(1<<i);
}
```

---

297. 二叉树的序列化与反序列化

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 (/faq/#binary-tree)。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。

先序遍历

一开始以为是prufer序列，但是考虑到编码和解码都可以用先序遍历，所以结果是不变的。

先序遍历时候遇到NULL就添加一个'!'，每个val值的结尾添加一个'?'。

前者是为了确定重建的边界条件，后者是为了把string类型的val值转化为int型。

注意重建的时候可以传一个引用下去，引用代表了现在遍历到了字符串的哪个位置，可以很方便的控制下标，如下所示：

```
        now->val=v;
        ++i;
        now->left=dfs(i,s);
        ++i;
        now->right=dfs(i,s);
        return now;
```

---

378. 有序矩阵中第K小的元素

给定一个 *n x n* 矩阵，其中每行和每列元素均按升序排序，找到矩阵中第 k 小的元素。
请注意，它是排序后的第 k 小元素，而不是第 k 个不同的元素。

示例:

```
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,

返回 13。
```

提示:
你可以假设 k 的值永远是有效的， $1 \le k \le n^2$ 。

水题，堆维护最小值。

一种朴素的想法是给每一行维护一个指针，每一次找到值最小的那一行，把那一行的指针向后移动一位。

答案就是最后最小的那个值。

不难发现找最小值可以用堆来维护，降低时间复杂度为O(klog(k))，实际上是O(n^2log(n))。

注意应当首先把k自减一次得到的答案才是我们想要的。

---

394. 字符串解码

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 1：

```
输入：s = "3[a]2[bc]"
输出："aaabcbc"
```

示例 2：

```
输入：s = "3[a2[c]]"
输出："accaccacc"
```

示例 3：

```
输入：s = "2[abc]3[cd]ef"
输出："abcabccdcdcdef"
```

示例 4：

```
输入：s = "abc3[cd]xyz"
输出："abccdcdcdxyz"
```

---

dfs

基本想法是做一个控制左边界 l 和右边界 r 的dfs，每次遇到方括号就递归地对方括号内的字符串 dfs 。

为了达到上述目的，首先要找到每一个左方括号对应的右方括号的位置， 可以使用 stack 从后往前遍历。遇到右方括号下标入栈，遇到左方括号出栈并且更新答案。

由于题目说明了字符串严格符合要求，即不会出现以下几种形式：

1. 3[4]
2. 2a
3. [a]

1 说明了方括号中没有数字， 2 规定了字符串的格式， 3 确保每一个方括号前面都有数字。

这样的话就直接遍历当前dfs的一段字符串，遇到普通字符就直接在答案中加上它，遇到数字就先把数字筛选出来再递归调用。参考代码如下：

```
string dfs(int l,int r) {
        // cout<<l<<' '<<r<<endl;
        string ret="";
        for(int i=l;i<=r;++i) {
            if(isdigit(s[i])) {
                int j=i;
                while(isdigit(s[j])) ++j;
                stringstream ss;
                int cnt=0;
                ss<<s.substr(i,j-i);
                ss>>cnt;
                string tmp=dfs(j+1,R[j]-1);
                while(cnt--) ret+=tmp;
                i=R[j];
            }
            else ret+=s[i];
        }
        return ret;
    }
```

399. 除法求值

给出方程式 `A / B = k`, 其中 `A` 和 `B` 均为用字符串表示的变量， `k` 是一个浮点型数字。根据已知方程式求解问题，并返回计算结果。如果结果不存在，则返回 `-1.0`。

示例 :
给定 `a / b = 2.0, b / c = 3.0`
问题: `a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?`
返回 `[6.0, 0.5, -1.0, 1.0, -1.0 ]`

输入为: `vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries` (方程式，方程式结果，问题方程式)， 其中 `equations.size() == values.size()`，即方程式的长度与方程式结果长度相等（程式与结果一一对应），并且结果值均为正数。以上为方程式的描述。 返回 `vector<double>` 类型。

基于上述例子，输入如下：

```
equations(方程式) = [ ["a", "b"], ["b", "c"] ],
values(方程式结果) = [2.0, 3.0],
queries(问题方程式) = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].
```

输入总是有效的。你可以假设除法运算中不会出现除数为0的情况，且不存在任何矛盾的结果。

带权并查集

每个点储存f[x]和val[x]，分别代表父亲节点和当前节点到父亲节点的权值。

带权并查集也可以使用路径压缩，代码如下：

```
    int _find(int x) {
            if(x==f[x]) return x;
            int tmp=_find(f[x]);
            val[x]*=val[f[x]];
            f[x]=tmp;
            return tmp;
    }
```

这里要注意路径压缩的条件是权值具有某种程度的可合并性，比如这道题中就可以把权值乘到一起而不改变最后的结果。

从x节点到y节点连一条权值为d的有向边的代码如下：

```
    void _merge(int x,int y,double d) {
        int rx=_find(x);
        int ry=_find(y);
        if(rx==ry) return;
        val[rx]=d*val[y]/val[x];
        f[rx]=ry;
    }
```

简单列一组式子就可以搞懂val[rx]是怎么算出来的了：

x-rx : val[x]

x-y : d

y-ry : val[y]

rx-ry : (x-y)+(y-ry)-(x-rx)

---

421. 数组中两个数的最大异或值

给定一个非空数组，数组中元素为 $a_0, a_1, a_2, \ldots , a_{n-1}$，其中 $0 \le a_i < 2^{31}$。

找到 a_i 和a_j 最大的异或 (XOR) 运算结果，其中$0 \le i, j < n$。

你能在O(n)的时间解决这个问题吗?

示例:

```
输入: [3, 10, 5, 25, 2, 8]

输出: 28

解释: 最大的结果是 5 ^ 25 = 28.
```

异或字典树

数字按从高位到低位插入，不用标记终点，因为插入长度都是一样的。

然后对每一个数字找最大异或值，能往另外一边走就走，并且累计答案，否则往相同边走。

442. 数组中重复的数据

给定一个整数数组 a，其中1 ≤ a[i] ≤ *n* （*n*为数组长度），其中有些元素出现**两次**而其他元素出现**一次**。

找到所有出现**两次**的元素。

你可以不用到任何额外空间并在O(*n*)时间复杂度内解决这个问题吗?

示例:

```
输入:
[4,3,2,7,8,2,3,1]

输出:
[2,3]
```

不使用额外空间的哈希，在不考虑正负数的情况下用正负号标记某数字是否出现过。

对于[1,n]，把它们映射到[0,n-1]，在进行判断的时候永远判断绝对值。

也就是说num[i]<0代表i已经出现过一次。

在遍历到num[i]时候如果发现对应的num[num[i]]已经小于0了，那么就把num[i]加入答案。

448. 找到所有数组中消失的数字

给定一个范围在 1 ≤ a[i] ≤ *n* (*n* = 数组大小) 的 整型数组，数组中的元素一些出现了两次，另一些只出现一次。

找到所有在 [1, *n*] 范围之间没有出现在数组中的数字。

您能在不使用额外空间且时间复杂度为*O(n)*的情况下完成这个任务吗? 你可以假定返回的数组不算在额外空间内。

示例:

```
输入:
[4,3,2,7,8,2,3,1]

输出:
[5,6]
```

联动：https://leetcode-cn.com/problems/find-all-duplicates-in-an-array/description/ (https://leetcode-cn.com/problems/find-all-duplicates-in-an-array/description/)

哈希的做法一模一样。

516. 最长回文子序列

给定一个字符串 `s` ，找到其中最长的回文子序列，并返回该序列的长度。可以假设 `s` 的最大长度为 `1000` 。

示例 1:
输入:

```
"bbbab"
```

输出:

```
4
```

一个可能的最长回文子序列为 "bbbb"。

示例 2:
输入:

```
"cbbd"
```

输出:

```
2
```

一个可能的最长回文子序列为 "bb"。

提示:

- `1 <= s.length <= 1000`
- `s` 只包含小写英文字母

DP，方程是： dp[i][j] = max(dp[i - 1][j] , dp[i][j - 1]); if(s[i - 1]==t[j - 1]) dp[i][j]=max(dp[i][j] , dp[i - 1][j - 1] + 1);

把原串倒过来在跟原来的串求最长公共子序列即可。

---

647. 回文子串

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，也会被计为是不同的子串。

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
说明: 6个回文子串: "a", "a", "a", "aa", "aa", "aaa".
```

注意:

1. 输入的字符串长度不会超过1000。

马拉车

统计答案的时候把单点半径除以二算到答案贡献上就行了，这样也可以排除新加入的 # 的影响。

---

648. 单词替换

在英语中，我们有一个叫做 词根 (root)的概念，它可以跟着其他一些词组成另一个较长的单词——我们称这个词为 继承词 (successor)。例如，词根 an ，跟随着单词 other (其他)，可以形成新的单词 another (另一个)。

现在，给定一个由许多词根组成的词典和一个句子。你需要将句子中的所有 继承词 用 词根 替换掉。如果 继承词 有许多可以形成它的 词根 ，则用最短的词根替换它。

你需要输出替换之后的句子。

示例：

```
输入：dict(词典) = ["cat", "bat", "rat"] sentence(句子) = "the cattle was rattled by the battery"
输出："the cat was rat by the bat"
```

提示：

- 输入只包含小写字母。
- 1 <= dict.length <= 1000
- 1 <= dict[i].length <= 100
- 1 <= 句中词语数 <= 1000
- 1 <= 句中词语长度 <= 1000

字典树。

把字典里的单词插进trie，每次用句子中的单词去trie中匹配，用第一个匹配位置替换句子中的单词即可。

cpp的split可以用stringstream。

力扣数组别开太大，否则会内存溢出。

---

699. 掉落的方块

在无限长的数轴（即 x 轴）上，我们根据给定的顺序放置对应的正方形方块。

第 i 个掉落的方块（positions[i] = (left, side_length)）是正方形，其中 left 表示该方块最左边的点位置(positions[i][0])，side_length 表示该方块的边长(positions[i][1])。

每个方块的底部边缘平行于数轴（即 x 轴），并且从一个比目前所有的落地方块更高的高度掉落而下。在上一个方块结束掉落，并保持静止后，才开始掉落新方块。

方块的底边具有非常大的粘性，并将保持固定在它们所接触的任何长度表面上（无论是数轴还是其他方块）。邻接掉落的边不会过早地粘合在一起，因为只有底边才具有粘性。

返回一个堆叠高度列表 ans 。每一个堆叠高度 ans[i] 表示在通过 positions[0], positions[1], ..., positions[i] 表示的方块掉落结束后，目前所有已经落稳的方块堆叠的最高高度。

示例 1:

```
输入: [[1, 2], [2, 3], [6, 1]]
输出: [2, 5, 5]
解释:

第一个方块 positions[0] = [1, 2] 掉落:
_aa
_aa
-------
方块最大高度为 2 。

第二个方块 positions[1] = [2, 3] 掉落:
__aaa
__aaa
__aaa
_aa__
_aa__
--------------
方块最大高度为5。
大的方块保持在较小的方块的顶部，不论它的重心在哪里，因为方块的底部边缘有非常大的粘性。

第三个方块 positions[1] = [6, 1] 掉落:
__aaa
__aaa
__aaa
_aa
_aa___a
--------------
方块最大高度为5。

因此，我们返回结果[2, 5, 5]。
```

示例 2:

```
输入: [[100, 100], [200, 100]]
输出: [100, 100]
解释: 相邻的方块不会过早地卡住，只有它们的底部边缘才能粘在表面上。
```

注意:

- `1 <= positions.length <= 1000`.
- `1 <= positions[i][0] <= 10^8`.
- `1 <= positions[i][1] <= 10^6`.

线段树

写两个操作，区间 set 和区间 max 查询。方块掉落等价于区间 set 成区间 max 加上当前方块的边长。

每次答案就是根节点的 max 。数据比较大，需要离散化一下。

---

718. 最长重复子数组

给两个整数数组 `A` 和 `B` ，返回两个数组中公共的、长度最长的子数组的长度。

示例:

```
输入：
A: [1,2,3,2,1]
B: [3,2,1,4,7]
输出：3
解释：
长度最长的公共子数组是 [3, 2, 1] 。
```

提示：

- `1 <= len(A), len(B) <= 1000`
- `0 <= A[i], B[i] < 100`

类似于最长公共子序列DP。

区别在于当某个位置上不同的时候就把这个位置的dp值置0，最后输出所有dp中的最大值即可，代码如下：

```
            if(A[i-1]==B[j-1])
                dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
            else
                dp[i][j]=0;
```

---

739. 每日温度

请根据每日 `气温` 列表，重新生成一个列表。对应位置的输出为：要想观测到更高的气温，至少需要等待的天数。如果气温在这之后都不会升高，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。

单调栈

要找到某个数字之后第一个大于它的数字，那么就正着扫一遍，维护一个递减的单调栈，每次栈顶出栈的时候更新答案。

---

771. 宝石与石头

给定字符串 `J` 代表石头中宝石的类型，和字符串 `S` 代表你拥有的石头。 `S` 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。

`J` 中的字母不重复， `J` 和 `S` 中的所有字符都是字母。字母区分大小写，因此 `"a"` 和 `"A"` 是不同类型的石头。

示例 1:

```
输入: J = "aA", S = "aAAbbbb"
输出: 3
```

示例 2:

```
输入: J = "z", S = "ZZ"
输出: 0
```

注意:

- `S` 和 `J` 最多含有50个字母。
- `J` 中的字符不重复。

水题， set 教学。

比较特别的是可以这样创建一个 set 来包含字符串 J 中的所有字符：

```
set<char> s(J.begin(),J.end());
```

有些时候做简单题非常好用。

---

780. 到达终点

从点 (x, y) 可以**转换**到 (x, x+y) 或者 (x+y, y)。

给定一个起点 (sx, sy) 和一个终点 (tx, ty)，如果通过一系列的**转换**可以从起点到达终点，则返回 True ，否则返回 False。

```
示例:
输入: sx = 1, sy = 1, tx = 3, ty = 5
输出: True
解释:
可以通过以下一系列转换从起点转换到终点:
(1, 1) -> (1, 2)
(1, 2) -> (3, 2)
(3, 2) -> (3, 5)

输入: sx = 1, sy = 1, tx = 2, ty = 2
输出: False

输入: sx = 1, sy = 1, tx = 1, ty = 1
输出: True
```

注意:

- sx, sy, tx, ty 是范围在 [1, 10^9] 的整数。

思维题。

考虑到数字都大于0，每一次操作一定会增加，那么不妨反向思考，每一次只要让较大的结果减去较小的结果就可以找到最初的情况。

为了加快运算速度，不用减很多次，每次减少的时候取余即可。取余可以保证每次都会导致大小交换。

考虑成功的条件，当有任意一个结果小于等于初始值时候就可以进行判断如下：

```
        if(tx==sx&&(ty-sy)%tx==0) return 1;
        if(ty==sy&&(tx-sx)%ty==0) return 1;
```

此外注意特判一下初始值大于结果的情况，因为上面式子中tx-sx和ty-sy可能小于0。

---

792. 匹配子序列的单词数

给定字符串 S 和单词字典 words，求 words[i] 中是 S 的子序列的单词个数。

```
示例:
输入:
S = "abcde"
words = ["a", "bb", "acd", "ace"]
输出: 3
解释: 有三个是 S 的子序列的单词: "a", "acd", "ace"。
```

注意:

- 所有在 words 和 S 里的单词都只由小写字母组成。
- S 的长度在 [1, 50000]。
- words 的长度在 [1, 5000]。
- words[i] 的长度在 [1, 50]。

---

sb优化。

子序列判断是O(n+m)的，为了避免对重复子序列的多次判断，做一个map来类似记忆化的方式保存答案。

---

805. 数组的均值分割

给定的整数数组 A，我们要将 A数组 中的每个元素移动到 B数组 或者 C数组中。（B数组和C数组在开始的时候都为空）

返回 `true` ，当且仅当在我们的完成这样的移动后，可使得B数组的平均值和C数组的平均值相等，并且B数组和C数组都不为空。

```
示例：
输入：
[1,2,3,4,5,6,7,8]
输出：true
解释：我们可以将数组分割为 [1,4,5,8] 和 [2,3,6,7]，他们的平均值都是4.5。
```

注意:

- `A` 数组的长度范围为 `[1, 30]`.
- `A[i]` 的数据范围为 `[0, 10000]`.

折半搜索

首先不难发现，部分平均数跟总体平均数应该是相等的。

考虑把数组分成两部分，前半部分和后半部分的平均数如果等于总体平均数，那么就找到了答案。

为了便于处理，我们把每个数字减去总体的平均数，那么我们只需要找到前后部分和为0的即算有解。

为了避开小数运算，可以在每个数字上乘以数组长度，不影响答案。

用map保存前半部分的和，后半部分进行以下三种检查：

1. 如果前半部分有和为0的，那么一定有解。
2. 不能同时选中前半部分和后半部分的全部元素，即使它们和为0。
3. 剩下的情况只需要判断后半部分和的相反数在map中是否存在。

---

837. 新21点

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：

1. `0 <= K <= N <= 10000`
2. `1 <= W <= 10000`
3. 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。
4. 此问题的判断限制时间已经减少。

概率DP，方程dp[i + j] += dp[i] * 1.0 / W

上面的方程写出来会超时，相当于用 O(n) 时间扫描， O(n) 时间求和。

那么考虑边算边更新 dp 的前缀和，代码如下：

```
        dp[0]=s[0]=1.0;
        for(int i=1;i<K;++i)
            dp[i]=1.0/W*sum(i-W,i-1),
            s[i]=dp[i]+s[i-1];
        for(int i=K;i<=N;++i)
            dp[i]=1.0/W*sum(i-W,K-1);
```

对于那些抵达后就停下的点就不需要再求前缀和了。

注意三个特判，如下：

```
        if(N>=K-1+W||K==0) return 1.0;
        if(N<K) return 0.0;
```

尤其是 K == 0 的情况，不特判的话下标会越界。

---

847. 访问所有节点的最短路径

给出 `graph` 为有 `N` 个节点（编号为 `0, 1, 2, ..., N-1`）的无向连通图。

`graph.length = N`，且只有节点 `i` 和 `j` 连通时，`j != i` 在列表 `graph[i]` 中恰好出现一次。

返回能够访问所有节点的最短路径的长度。你可以在任一节点开始和停止，也可以多次重访节点，并且可以重用边。

示例 1：

```
输入：[[1,2,3],[0],[0],[0]]
输出：4
解释：一个可能的路径为 [1,0,2,0,3]
```

示例 2：

```
输入：[[1],[0,2,4],[1,3,4],[2],[1,2]]
输出：4
解释：一个可能的路径为 [0,1,4,2,3]
```

提示：

1. `1 <= graph.length <= 12`
2. `0 <= graph[i].length < graph.length`

二进制bfs

用二进制表示点是否访问， queue 里放当前状态和抵达了哪个点，复杂度 O(2^n*n)

一开始认为是在最小生成树上做手脚，没想到完全跑偏了，但是总感觉数据大一点应该就是最小生成树上的操作了。

---

928. 尽量减少恶意软件的传播 II

(这个问题与 *尽量减少恶意软件的传播* 是一样的，不同之处用粗体表示。)

在节点网络中，只有当 `graph[i][j] = 1` 时，每个节点 `i` 能够直接连接到另一个节点 `j`。

一些节点 `initial` 最初被恶意软件感染。只要两个节点直接连接，且其中至少一个节点受到恶意软件的感染，那么两个节点都将被恶意软件感染。这种恶意软件的传播将继续，直到没有更多的节点可以被这种方式感染。

假设 `M(initial)` 是在恶意软件停止传播之后，整个网络中感染恶意软件的最终节点数。

我们可以从初始列表中删除一个节点，**并完全移除该节点以及从该节点到任何其他节点的任何连接。**如果移除这一节点将最小化 `M(initial)`， 则返回该节点。如果有多个节点满足条件，就返回索引最小的节点。

示例 1：

```
输出：graph = [[1,1,0],[1,1,0],[0,0,1]], initial = [0,1]
输入：0
```

示例 2：

```
输入：graph = [[1,1,0],[1,1,1],[0,1,1]], initial = [0,1]
输出：1
```

示例 3：

```
输入：graph = [[1,1,0,0],[1,1,1,0],[0,1,1,1],[0,0,1,1]], initial = [0,1]
输出：1
```

提示：

1. `1 < graph.length = graph[0].length <= 300`
2. `0 <= graph[i][j] == graph[j][i] <= 1`
3. `graph[i][i] = 1`
4. `1 <= initial.length < graph.length`
5. `0 <= initial[i] < graph.length`

并查集。

考虑到数据范围比较小，我们枚举割点，用O(n^2)重建图。然后利用并查集判断连通块大小更新答案即可。复杂度O(n^3)。

但是这样复杂度太吃瘪了，数据稍微大一点就会翻车。

应该这样考虑，我们把未被感染的节点合并到一起，计算一下它的度数。

如果它的度数为1，说明有某个感染节点被删除后就会解放这一块；如果度数不为1，那么说明不管怎样删除都不会解放这一块。

接下来我们只要对每个点O(n)统计它能解放哪一块就可以了。总复杂度就降到了O(n^2)。

---

974. 和可被 K 整除的子数组

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. 1 <= A.length <= 30000
2. -10000 <= A[i] <= 10000
3. 2 <= K <= 10000

前缀和与map

考虑做一个前缀和，由于题目要求和为k的倍数的子数组个数，我们只考虑前缀和对k取余后的结果。初步处理如下：

```
for(int i=1;i<A.size();++i) A[i]+=A[i-1];
for(auto &i:A) i%=K;
```

接下来考虑如何统计答案。如果有两个不同的位置上前缀和相等，那么这一段数组之和对k取余一定等于 0 。所以每次我们先统计答案让 ans += map[i]，然后把 map 中对应前缀和的值加 1 。代码如下：

```
ans+=mp[i];
mp[i]++;
```

但是这只是第一部分答案，由于前缀和可能有负数，所以要考虑前缀和符号不同导致的差为 k 的倍数的答案。

```
if(i<0) ans+=mp[i+K];
else ans+=mp[i-K];
```

还没完，对于那些前缀和等于 0 的位置，它们不需要与任何数字组合，自己就能构成答案，所以要再额外统计一次。

```
for(auto i:A) ans+=(i==0);
```

---

983. 最低票价

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. `1 <= days.length <= 365`
2. `1 <= days[i] <= 365`
3. `days` 按顺序严格递增
4. `costs.length == 3`
5. `1 <= costs[i] <= 1000`

DP，方程：

dp[i] = min(dp[i] , dp[i - 1] + costs[0]);

dp[j] = min(dp[j] , dp[i - 1] + costs[1]); j >= i && j <= i + 6

dp[j] = min(dp[j] , dp[i - 1] + costs[2]); j >= i && j <= i + 29

题面比较怪，注意对那些不在考虑范围之内的日期令：dp[i] = dp[i - 1] 就好。

找出不在范围内的日期可以 O(nlogn) 排序之后直接用单指针 O(n) 找出来。

dp 的范围是 365 天。

---

990. 等式方程的可满足性

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：

```
输入：["b==a","a==b"]
输出：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. `1 <= equations.length <= 500`
2. `equations[i].length == 4`
3. `equations[i][0]` 和 `equations[i][3]` 是小写字母
4. `equations[i][1]` 要么是 `'='`，要么是 `'!'`
5. `equations[i][2]` 是 `'='`

并查集

相等的在一棵树下，不相等的判断一下在不在同一棵树下就OK了。

---

999. 可以被一步捕获的棋子数

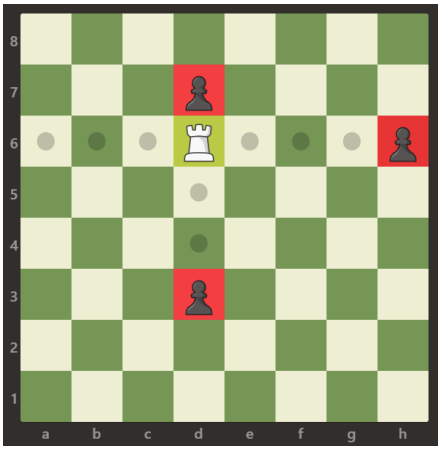
在一个 8 x 8 的棋盘上，有一个白色的车（Rook），用字符 'R' 表示。棋盘上还可能存在空方块，白色的象（Bishop）以及黑色的卒（pawn），分别用字符 '.'， 'B' 和 'p' 表示。不难看出，大写字符表示的是白棋，小写字符表示的是黑棋。

车按国际象棋中的规则移动。东，西，南，北四个基本方向任选其一，然后一直向选定的方向移动，直到满足下列四个条件之一：

- 棋手选择主动停下来。
- 棋子因到达棋盘的边缘而停下。
- 棋子移动到某一方格来捕获位于该方格上敌方（黑色）的卒，停在该方格内。
- 车不能进入/越过已经放有其他友方棋子（白色的象）的方格，停在友方棋子前。

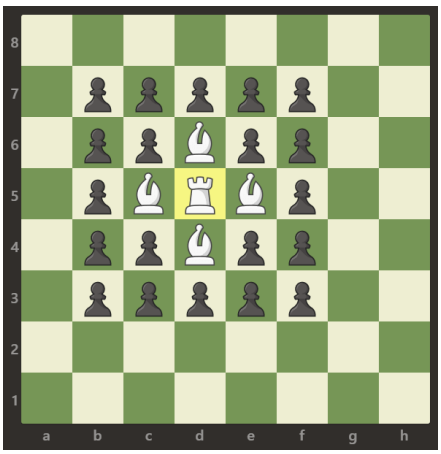
你现在可以控制车移动一次，请你统计有多少敌方的卒处于你的捕获范围内（即，可以被一步捕获的棋子数）。

示例 1：



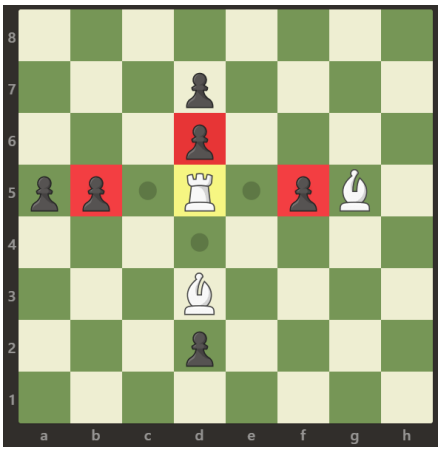
```
输入：[[".",".",".",".",".",".",".",".","."],[".",".",".","p",".",".",".",".","."],[".",".",".","R",".",".",".","p"],[".",".",".",
输出：3
解释：
在本例中，车能够捕获所有的卒。
```

示例 2：



```
输入：[[".",".",".",".",".",".",".",".","."],[".","p","p","p","p","p",".","."],[".","p","p","B","p","p",".","."],[".","p","B
输出：0
解释：
象阻止了车捕获任何卒。
```

示例 3：



```
输入：[[".",".",".",".",".",".",".",".","."],[".",".",".","p",".",".",".",".","."],[".",".",".","p",".",".",".",".","."],["p","p",".
输出：3
解释：
车可以捕获位置 b5，d6 和 f5 的卒。
```

提示：

1. board.length == board[i].length == 8
2. board[i][j] 可以是 'R'，'.'，'B' 或 'p'
3. 只有一个格子上存在 board[i][j] == 'R'

水题，四个方向搜一遍即可。

1014. 最佳观光组合

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：

1. 2 <= A.length <= 50000
2. 1 <= A[i] <= 1000

思维题。

把式子转换一下可以发现等于A[i]+i+A[j]-j，其中i>j。

那么每一次算到一个新的i就更新答案，再记录一下A[i]-i的最大值即可。

1028. 从先序遍历还原二叉树

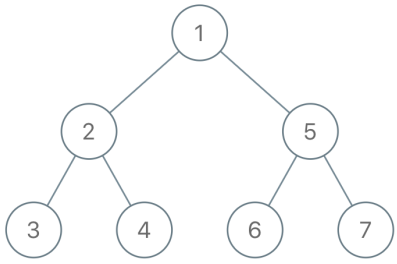
我们从二叉树的根节点 `root` 开始进行深度优先搜索。

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。

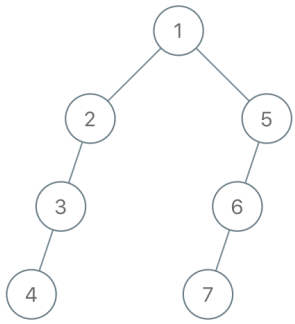
给出遍历输出 `S`，还原树并返回其根节点 `root`。

示例 1：



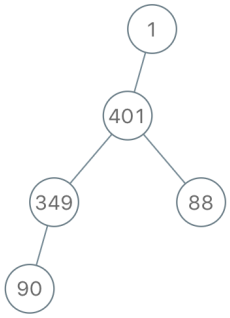

```
输入："1-2--3--4-5--6--7"
输出：[1,2,5,3,4,6,7]
```

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。

dfs。

做两个函数用来获取深度和数字，判断下一层的深度是否等于当前深度加一，如果等于就向下dfs。

注意传引用参数，控制当前i在原字符串s中的位置。此外dfs也传一个记录深度的参数下去。

注意判断下一层的深度如果不等于当前深度，要把下标退回去，如下：

```
            --i;
            while(s[i]=='-') --i;
            ++i;
```

1160. 拼写单词

给你一份『词汇表』（字符串数组） words 和一张『字母表』（字符串） chars 。

假如你可以用 chars 中的『字母』（字符）拼写出 words 中的某个『单词』（字符串），那么我们就认为你掌握了这个单词。

注意：每次拼写（指拼写词汇表中的一个单词）时， chars 中的每个字母都只能用一次。

返回词汇表 words 中你掌握的所有单词的 **长度之和。**

示例 1：

```
输入：words = ["cat","bt","hat","tree"], chars = "atach"
输出：6
解释：
可以形成字符串 "cat" 和 "hat"，所以答案是 3 + 3 = 6。
```

示例 2：

```
输入：words = ["hello","world","leetcode"], chars = "welldonehoneyr"
输出：10
解释：
可以形成字符串 "hello" 和 "world"，所以答案是 5 + 5 = 10。
```

提示：

1. 1 <= words.length <= 1000
2. 1 <= words[i].length, chars.length <= 100
3. 所有字符串中都仅包含小写英文字母

水题，做个数组当 map 即可， char 范围比较小，几乎可以做到 O(1) 查询。

1300. 转变数组后最接近目标值的数组和

给你一个整数数组 arr 和一个目标值 target ，请你返回一个整数 value ，使得将数组中所有大于 value 的值变成 value 后，数组的和最接近 target （最接近表示两者之差的绝对值最小）。

如果有多种使得和最接近 target 的方案，请你返回这些整数中的最小值。

请注意，答案不一定是 arr 中的数字。

示例 1：

```
输入：arr = [4,9,3], target = 10
输出：3
解释：当选择 value 为 3 时，数组会变成 [3, 3, 3]，和为 9 ，这是最接近 target 的方案。
```

示例 2：

```
输入：arr = [2,3,5], target = 10
输出：5
```

示例 3：

```
输入：arr = [60864,25176,27249,21296,20204], target = 56803
输出：11361
```

提示：

- `1 <= arr.length <= 10^4`
- `1 <= arr[i], target <= 10^5`

神秘二分

很容易想到二分值域，每次 O(n) 来 check 。答案大于等于了就调整r，小于了就调整l。

但是这样有个大问题，对于一些很大的数字， check 的结果总是 ok 的，所以需要规定二分上限是数列的 max 。

下限不用管，因为我们总是希望答案更小。

此外，二分结束之后还要判断 l 和 r 的值哪个更接近，返回更接近的那个即可。

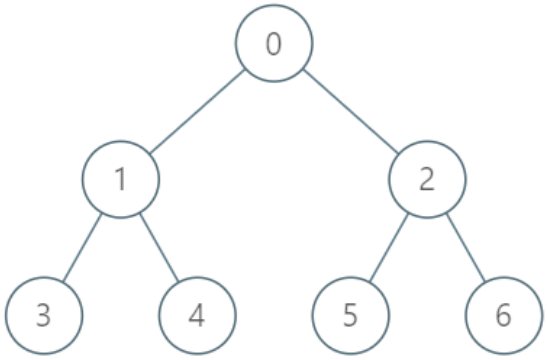
1483. 树节点的第 K 个祖先

给你一棵树，树上有 `n` 个节点，按从 `0` 到 `n-1` 编号。树以父节点数组的形式给出，其中 `parent[i]` 是节点 `i` 的父节点。树的根节点是编号为 `0` 的节点。

请你设计并实现 `getKthAncestor(int node, int k)` 函数，函数返回节点 `node` 的第 `k` 个祖先节点。如果不存在这样的祖先节点，返回 `-1` 。

树节点的第 *k* 个祖先节点是从该节点到根节点路径上的第 `k` 个节点。

示例：




```
输入：
["TreeAncestor","getKthAncestor","getKthAncestor","getKthAncestor"]
[[7,[-1,0,0,1,1,2,2]],[3,1],[5,2],[6,3]]

输出：
[null,1,0,-1]

解释：
TreeAncestor treeAncestor = new TreeAncestor(7, [-1, 0, 0, 1, 1, 2, 2]);

treeAncestor.getKthAncestor(3, 1);  // 返回 1 ，它是 3 的父节点
treeAncestor.getKthAncestor(5, 2);  // 返回 0 ，它是 5 的祖父节点
treeAncestor.getKthAncestor(6, 3);  // 返回 -1 因为不存在满足要求的祖先节点
```

提示：

- `1 <= k <= n <= 5*10^4`
- `parent[0] == -1` 表示编号为 `0` 的节点是根节点。
- 对于所有的 `0 < i < n` ， `0 <= parent[i] < n` 总成立
- `0 <= node < n`
- 至多查询 `5*10^4` 次

倍增

跟lca想法一模一样。

1202. 交换字符串中的元素

给你一个字符串 `s`，以及该字符串中的一些「索引对」数组 `pairs`，其中 `pairs[i] = [a, b]` 表示字符串中的两个索引（编号从 0 开始）。

你可以 **任意多次交换** 在 `pairs` 中任意一对索引处的字符。

返回在经过若干次交换后，`s` 可以变成的按字典序最小的字符串。

示例 1:

```
输入：s = "dcab", pairs = [[0,3],[1,2]]
输出："bacd"
解释：
交换 s[0] 和 s[3], s = "bcad"
交换 s[1] 和 s[2], s = "bacd"
```

示例 2:

```
输入：s = "dcab", pairs = [[0,3],[1,2],[0,2]]
输出："abcd"
解释：
交换 s[0] 和 s[3], s = "bcad"
交换 s[0] 和 s[2], s = "acbd"
交换 s[1] 和 s[2], s = "abcd"
```

示例 3:

```
输入：s = "cba", pairs = [[0,1],[1,2]]
输出："abc"
解释：
交换 s[0] 和 s[1], s = "bca"
交换 s[1] 和 s[2], s = "bac"
交换 s[0] 和 s[1], s = "abc"
```

提示:

- `1 <= s.length <= 10^5`
- `0 <= pairs.length <= 10^5`
- `0 <= pairs[i][0], pairs[i][1] < s.length`
- `s` 中只含有小写英文字母

并查集

把能互相交换到的字符放在一棵树下，把一棵树下所有字符按字典序排序，从小到大填上即可。

1250. 检查「好数组」

给你一个正整数数组 `nums`，你需要从中任选一些子集，然后将子集中每一个数乘以一个 **任意整数**，并求出他们的和。

假如该和结果为 `1`，那么原数组就是一个「**好数组**」，则返回 `True`；否则请返回 `False`。

示例 1：

```
输入：nums = [12,5,7,23]
输出：true
解释：挑选数字 5 和 7。
5*3 + 7*(-2) = 1
```

示例 2：

```
输入：nums = [29,6,10]
输出：true
解释：挑选数字 29, 6 和 10。
29*1 + 6*(-3) + 10*(-1) = 1
```

示例 3：

```
输入：nums = [3,6]
输出：false
```

提示：

- 1 <= nums.length <= 10^5
- 1 <= nums[i] <= 10^9

思维题

大概猜个结论，如果有两个数字互质那么这两个数字就能成为“好数组”。

于是求一边整个数组的gcd，如果为1的话返回true。

1342. 将数字变成 0 的操作次数

给你一个非负整数 num ，请你返回将它变成 0 所需要的步数。 如果当前数字是偶数，你需要把它除以 2 ；否则，减去 1 。

示例 1：

```
输入：num = 14
输出：6
解释：
步骤 1) 14 是偶数，除以 2 得到 7 。
步骤 2） 7 是奇数，减 1 得到 6 。
步骤 3） 6 是偶数，除以 2 得到 3 。
步骤 4） 3 是奇数，减 1 得到 2 。
步骤 5） 2 是偶数，除以 2 得到 1 。
步骤 6） 1 是奇数，减 1 得到 0 。
```

示例 2：

```
输入：num = 8
输出：4
解释：
步骤 1） 8 是偶数，除以 2 得到 4 。
步骤 2） 4 是偶数，除以 2 得到 2 。
步骤 3） 2 是偶数，除以 2 得到 1 。
步骤 4） 1 是奇数，减 1 得到 0 。
```

示例 3：

```
输入：num = 123
输出：12
```

提示：

- `0 <= num <= 10^6`

水题，按照题意dfs即可，复杂度在 O(log(n)) 级别。

1368. 使网格图至少有一条有效路径的最小代价

给你一个 m x n 的网格图 `grid` 。 `grid` 中每个格子都有一个数字，对应着从该格子出发下一步走的方向。 `grid[i][j]` 中的数字可能为以下几种情况：

- **1** ，下一步往右走，也就是你会从 `grid[i][j]` 走到 `grid[i][j + 1]`
- **2** ，下一步往左走，也就是你会从 `grid[i][j]` 走到 `grid[i][j - 1]`
- **3** ，下一步往下走，也就是你会从 `grid[i][j]` 走到 `grid[i + 1][j]`
- **4** ，下一步往上走，也就是你会从 `grid[i][j]` 走到 `grid[i - 1][j]`

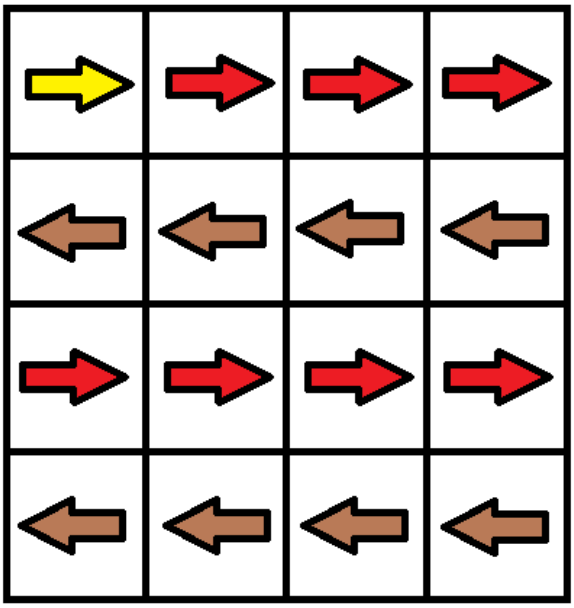
注意网格图中可能会有 **无效数字** ，因为它们可能指向 `grid` 以外的区域。

一开始，你会从最左上角的格子 `(0,0)` 出发。我们定义一条 **有效路径** 为从格子 `(0,0)` 出发，每一步都顺着数字对应方向走，最终在最右下角的格子 `(m - 1, n - 1)` 结束的路径。有效路径 **不需要是最短路径** 。

你可以花费 `cost = 1` 的代价修改一个格子中的数字，但每个格子中的数字 **只能修改一次** 。

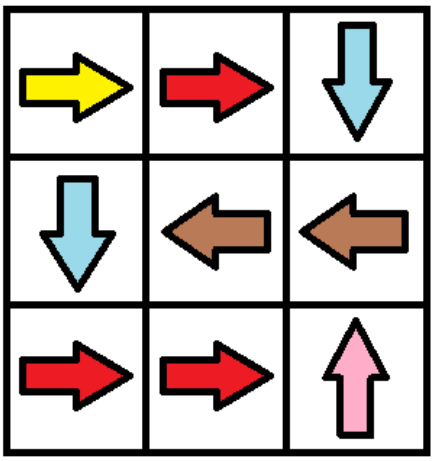
请你返回让网格图至少有一条有效路径的最小代价。

示例 1：



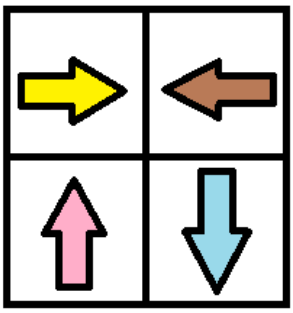
```
输入：grid = [[1,1,1,1],[2,2,2,2],[1,1,1,1],[2,2,2,2]]
输出：3
解释：你将从点 (0, 0) 出发。
到达 (3, 3) 的路径为： (0, 0) --> (0, 1) --> (0, 2) --> (0, 3) 花费代价 cost = 1 使方向向下 --> (1, 3) --> (1, 2) --> (1, 1
总花费为 cost = 3.
```

示例 2：



```
输入：grid = [[1,1,3],[3,2,2],[1,1,4]]
输出：0
解释：不修改任何数字你就可以从 (0, 0) 到达 (2, 2) 。
```

示例 3：



```
输入：grid = [[1,2],[4,3]]
输出：1
```

示例 4：

```
输入：grid = [[2,2,2],[2,2,2]]
输出：3
```

示例 5：

```
输入：grid = [[4]]
输出：0
```

提示：

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 100`

最短路

建一张四联通网格图，某个格点一开始指向的方向边权为 0 ， 其余边权为 1 ， 跑一边最短路即可。

1431. 拥有最多糖果的孩子

给你一个数组 `candies` 和一个整数 `extraCandies` ，其中 `candies[i]` 代表第 `i` 个孩子拥有的糖果数目。

对每一个孩子，检查是否存在一种方案，将额外的 `extraCandies` 个糖果分配给孩子们之后，此孩子有 **最多** 的糖果。注意，允许有多个孩子同时拥有 **最多** 的糖果数目。

示例 1：

```
输入：candies = [2,3,5,1,3], extraCandies = 3
输出：[true,true,true,false,true]
解释：
孩子 1 有 2 个糖果，如果他得到所有额外的糖果（3个），那么他总共有 5 个糖果，他将成为拥有最多糖果的孩子。
孩子 2 有 3 个糖果，如果他得到至少 2 个额外糖果，那么他将成为拥有最多糖果的孩子。
孩子 3 有 5 个糖果，他已经是拥有最多糖果的孩子。
孩子 4 有 1 个糖果，即使他得到所有额外的糖果，他也只有 4 个糖果，无法成为拥有糖果最多的孩子。
孩子 5 有 3 个糖果，如果他得到至少 2 个额外糖果，那么他将成为拥有最多糖果的孩子。
```

示例 2：

```
输入：candies = [4,2,1,1,2], extraCandies = 1
输出：[true,false,false,false,false]
解释：只有 1 个额外糖果，所以不管额外糖果给谁，只有孩子 1 可以成为拥有糖果最多的孩子。
```

示例 3：

```
输入：candies = [12,1,12], extraCandies = 10
输出：[true,false,true]
```

提示：

- `2 <= candies.length <= 100`
- `1 <= candies[i] <= 100`
- `1 <= extraCandies <= 50`

水题，找到最大值遍历一遍即可。

1455. 检查单词是否为句中其他单词的前缀

给你一个字符串 `sentence` 作为句子并指定检索词为 `searchWord` ，其中句子由若干用 **单个空格** 分隔的单词组成。

请你检查检索词 `searchWord` 是否为句子 `sentence` 中任意单词的前缀。

- 如果 `searchWord` 是某一个单词的前缀，则返回句子 `sentence` 中该单词所对应的下标（**下标从 1 开始**）。
- 如果 `searchWord` 是多个单词的前缀，则返回匹配的第一个单词的下标（**最小下标**）。
- 如果 `searchWord` 不是任何单词的前缀，则返回 **-1** 。

字符串 `S` 的「前缀」是 `S` 的任何前导连续子字符串。

示例 1：

```
输入：sentence = "i love eating burger", searchWord = "burg"
输出：4
解释："burg" 是 "burger" 的前缀，而 "burger" 是句子中第 4 个单词。
```

示例 2：

```
输入：sentence = "this problem is an easy problem", searchWord = "pro"
输出：2
解释："pro" 是 "problem" 的前缀，而 "problem" 是句子中第 2 个也是第 6 个单词，但是应该返回最小下标 2 。
```

示例 3：

```
输入：sentence = "i am tired", searchWord = "you"
输出：-1
解释："you" 不是句子中任何单词的前缀。
```

示例 4：

```
输入：sentence = "i use triple pillow", searchWord = "pill"
输出：4
```

示例 5：

```
输入：sentence = "hello from the other side", searchWord = "they"
输出：-1
```

提示：

- `1 <= sentence.length <= 100`
- `1 <= searchWord.length <= 10`
- `sentence` 由小写英文字母和空格组成。
- `searchWord` 由小写英文字母组成。
- 前缀就是紧密附着于词根的语素，中间不能插入其它成分，并且它的位置是固定的——-位于词根之前。（引用自 前缀_百度百科 (https://baike.baidu.com/item/%E5%89%8D%E7%BC%80)）

---

stringstream教学

想要在cpp中实现split()函数，可以把标识符设置为空格，然后使用下面的代码：

```
stringstream ss;
ss<<s;
ss>>s;
do {
    //work(s)
}while(ss>>s);
```

work()函数可以随意改写，这里写成把v.push_back(s)就行。最后获得的v就是所有单词构成的数组。

1456. 定长子串中元音的最大数目

给你字符串 `s` 和整数 `k` 。

请返回字符串 `s` 中长度为 `k` 的单个子字符串中可能包含的最大元音字母数。

英文中的 **元音字母** 为（`a`，`e`，`i`，`o`，`u`）。

示例 1：

```
输入：s = "abciiidef", k = 3
输出：3
解释：子字符串 "iii" 包含 3 个元音字母。
```

示例 2：

```
输入：s = "aeiou", k = 2
输出：2
解释：任意长度为 2 的子字符串都包含 2 个元音字母。
```

示例 3：

```
输入：s = "leetcode", k = 3
输出：2
解释："lee"、"eet" 和 "ode" 都包含 2 个元音字母。
```

示例 4：

```
输入：s = "rhythms", k = 4
输出：0
解释：字符串 s 中不含任何元音字母。
```

示例 5：

```
输入：s = "tryhard", k = 4
输出：1
```

提示：

- `1 <= s.length <= 10^5`
- `s` 由小写英文字母组成
- `1 <= k <= s.length`

前缀和

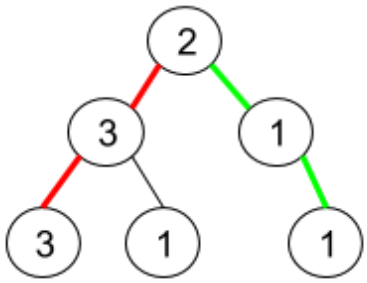
遍历一遍字符串，做一个关于元音字母的前缀和，然后再遍历一遍字符串，用 i 和 i - k 更新答案。

1457. 二叉树中的伪回文路径

给你一棵二叉树，每个节点的值为 1 到 9 。我们称二叉树中的一条路径是 「**伪回文**」的，当它满足：路径经过的所有节点值的排列中，存在一个回文序列。

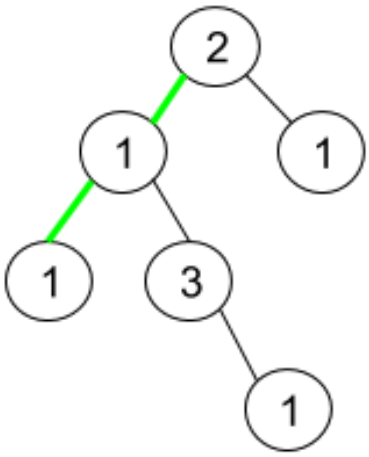
请你返回从根到叶子节点的所有路径中 **伪回文** 路径的数目。

示例 1：




```
输入：root = [2,3,1,3,1,null,1]
输出：2
解释：上图为给定的二叉树。总共有 3 条从根到叶子的路径：红色路径 [2,3,3] ，绿色路径 [2,1,1] 和路径 [2,3,1] 。
     在这些路径中，只有红色和绿色的路径是伪回文路径，因为红色路径 [2,3,3] 存在回文排列 [3,2,3] ，绿色路径 [2,1,1] 存在回文排列 [
```

示例 2：




```
输入：root = [2,1,1,1,3,null,null,null,null,null,1]
输出：1
解释：上图为给定二叉树。总共有 3 条从根到叶子的路径：绿色路径 [2,1,1] ，路径 [2,1,3,1] 和路径 [2,1] 。
     这些路径中只有绿色路径是伪回文路径，因为 [2,1,1] 存在回文排列 [1,2,1] 。
```

示例 3：

```
输入：root = [9]
输出：1
```

提示：

- 给定二叉树的节点数目在 1 到 10^5 之间。
- 节点值在 1 到 9 之间。

---

map加dfs

回文串的特色是出现奇数次的字符要么没有要么只有一个，用map记录出现次数，dfs搜到叶子节点更新答案就可以。

1458. 两个子序列的最大点积

给你两个数组 nums1 和 nums2 。

请你返回 nums1 和 nums2 中两个长度相同的 **非空** 子序列的最大点积。

数组的非空子序列是通过删除原数组中某些元素（可能一个也不删除）后剩余数字组成的序列，但不能改变数字间相对顺序。比方说，[2,3,5] 是 [1,2,3,4,5] 的一个子序列而 [1,5,3] 不是。

示例 1：

```
输入：nums1 = [2,1,-2,5], nums2 = [3,0,-6]
输出：18
解释：从 nums1 中得到子序列 [2,-2] ，从 nums2 中得到子序列 [3,-6] 。
它们的点积为 (2*3 + (-2)*(-6)) = 18 。
```

示例 2：

```
输入：nums1 = [3,-2], nums2 = [2,-6,7]
输出：21
解释：从 nums1 中得到子序列 [3] ，从 nums2 中得到子序列 [7] 。
它们的点积为 (3*7) = 21 。
```

示例 3：

```
输入：nums1 = [-1,-1], nums2 = [1,1]
输出：-1
解释：从 nums1 中得到子序列 [-1] ，从 nums2 中得到子序列 [1] 。
它们的点积为 -1 。
```

提示：

- 1 <= nums1.length, nums2.length <= 500
- -1000 <= nums1[i], nums2[i] <= 100

点积：

定义 **a** = [a_1, a_2,…, a_n] 和 **b** = [b_1, b_2,…, b_n] 的点积为：

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

这里的 **Σ** 指示总和符号。

DP，方程是： dp[i][j]=max{ dp[i - 1][j] , dp[i][j - 1] , dp[i - 1][j - 1] + a[i] * b[j] }

很类似于最长公共子序列的方程。实际写代码的时候主要考虑以下两个问题：

1. i 和 j 的遍历范围最好是 [1 , n] 和 [1 , m] ， 这样的话可以免去dp下标越界的问题。但是相对应的必须修改数组 a 和 b 对应的下标以防止数组下标出界。
2. 这道题有一种特殊情况： a 和 b 数组一个全为正一个全为负，此时dp的答案会是 0 。因为dp的初值就是 0 ， 永远大于其他负数。特判一下即可。

1475. 商品折扣后的最终价格

给你一个数组 prices ，其中 prices[i] 是商店里第 i 件商品的价格。

商店里正在进行促销活动，如果你要买第 i 件商品，那么你可以得到与 prices[j] 相等的折扣，其中 j 是满足 j > i 且 prices[j] <= prices[i] 的 **最小下标** ，如果没有满足条件的 j ，你将没有任何折扣。

请你返回一个数组，数组中第 i 个元素是折扣后你购买商品 i 最终需要支付的价格。

示例 1：

```
输入：prices = [8,4,6,2,3]
输出：[4,2,4,2,3]
解释：
商品 0 的价格为 price[0]=8 ，你将得到 prices[1]=4 的折扣，所以最终价格为 8 - 4 = 4 。
商品 1 的价格为 price[1]=4 ，你将得到 prices[3]=2 的折扣，所以最终价格为 4 - 2 = 2 。
商品 2 的价格为 price[2]=6 ，你将得到 prices[3]=2 的折扣，所以最终价格为 6 - 2 = 4 。
商品 3 和 4 都没有折扣。
```

示例 2：

```
输入：prices = [1,2,3,4,5]
输出：[1,2,3,4,5]
解释：在这个例子中，所有商品都没有折扣。
```

示例 3：

```
输入：prices = [10,1,1,6]
输出：[9,0,1,6]
```

提示：

- `1 <= prices.length <= 500`
- `1 <= prices[i] <= 10^3`

水题。

O(n^2)算一下就行，注意折扣是减少一部分价格。

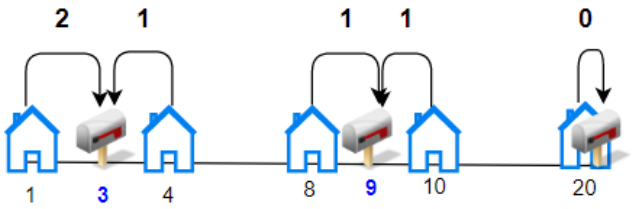
1478. 安排邮筒

给你一个房屋数组 `houses` 和一个整数 `k` ，其中 `houses[i]` 是第 `i` 栋房子在一条街上的位置，现需要在这条街上安排 `k` 个邮筒。

请你返回每栋房子与离它最近的邮筒之间的距离的 **最小** 总和。

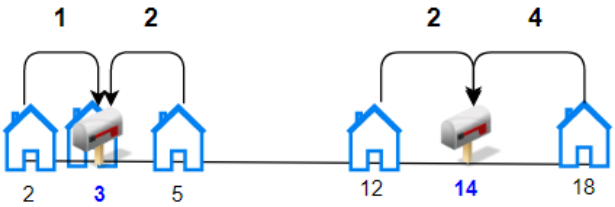
答案保证在 32 位有符号整数范围以内。

示例 1：




```
输入：houses = [1,4,8,10,20], k = 3
输出：5
解释：将邮筒分别安放在位置 3， 9 和 20 处。
每个房子到最近邮筒的距离和为 |3-1| + |4-3| + |9-8| + |10-9| + |20-20| = 5 。
```

示例 2：




```
输入：houses = [2,3,5,12,18], k = 2
输出：9
解释：将邮筒分别安放在位置 3 和 14 处。
每个房子到最近邮筒距离和为 |2-3| + |3-3| + |5-3| + |12-14| + |18-14| = 9 。
```

示例 3：

```
输入：houses = [7,4,6,1], k = 1
输出：8
```

示例 4：

```
输入：houses = [3,6,14,10], k = 4
输出：0
```

提示：

- `n == houses.length`
- `1 <= n <= 100`
- `1 <= houses[i] <= 10^4`
- `1 <= k <= n`

- 数组 houses 中的整数互不相同。

---

dp。方程dp[i][j]=min(dp[i][j],dp[l][j-1]+mid[l+1][i])。

一维点的曼哈顿距离最小值在一维点的中位数位置。

题目就转化成了把n个点分成k段，每一段都取中位数位置，最小化总代价。

mid是可以预处理的，由于中位数只能放在整点上。所以对于偶数长度的段，需要判断最接近中位数的两个整点的答案，取小的那个。

1476. 子矩形查询

请你实现一个类 SubrectangleQueries ，它的构造函数的参数是一个 rows x cols 的矩形（这里用整数矩阵表示），并支持以下两种操作：

1. updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)
   - 用 newValue 更新以 (row1,col1) 为左上角且以 (row2,col2) 为右下角的子矩形。
2. getValue(int row, int col)
   - 返回矩形中坐标 (row,col) 的当前值。

示例 1：

```
输入：
["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle","getValue","getValue"]
[[[[1,2,1],[4,3,4],[3,2,1],[1,1,1]]],[0,2],[0,0,3,2,5],[0,2],[3,1],[3,0,3,2,10],[3,1],[0,2]]
输出：
[null,1,null,5,5,null,10,5]
解释：
SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,2,1],[4,3,4],[3,2,1],[1,1,1]]);
// 初始的 (4x3) 矩形如下：
// 1 2 1
// 4 3 4
// 3 2 1
// 1 1 1
subrectangleQueries.getValue(0, 2); // 返回 1
subrectangleQueries.updateSubrectangle(0, 0, 3, 2, 5);
// 此次更新后矩形变为：
// 5 5 5
// 5 5 5
// 5 5 5
// 5 5 5
subrectangleQueries.getValue(0, 2); // 返回 5
subrectangleQueries.getValue(3, 1); // 返回 5
subrectangleQueries.updateSubrectangle(3, 0, 3, 2, 10);
// 此次更新后矩形变为：
// 5   5   5
// 5   5   5
// 5   5   5
// 10  10  10
subrectangleQueries.getValue(3, 1); // 返回 10
subrectangleQueries.getValue(0, 2); // 返回 5
```

示例 2：

```
输入：
["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle","getValue"]
[[[[1,1,1],[2,2,2],[3,3,3]]],[0,0],[0,0,2,2,100],[0,0],[2,2],[1,1,2,2,20],[2,2]]
输出：
[null,1,null,100,100,null,20]
解释：
SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,1,1],[2,2,2],[3,3,3]]);
subrectangleQueries.getValue(0, 0); // 返回 1
subrectangleQueries.updateSubrectangle(0, 0, 2, 2, 100);
subrectangleQueries.getValue(0, 0); // 返回 100
subrectangleQueries.getValue(2, 2); // 返回 100
subrectangleQueries.updateSubrectangle(1, 1, 2, 2, 20);
subrectangleQueries.getValue(2, 2); // 返回 20
```

提示：

- 最多有 500 次 updateSubrectangle 和 getValue 操作。
- 1 <= rows, cols <= 100
- rows == rectangle.length
- cols == rectangle[i].length
- 0 <= row1 <= row2 < rows
- 0 <= col1 <= col2 < cols
- 1 <= newValue, rectangle[i][j] <= 10^9
- 0 <= row < rows
- 0 <= col < cols

水题。

O(n^2)更新即可。

1477. 找两个和为目标值且不重叠的子数组

给你一个整数数组 arr 和一个整数值 target 。

请你在 arr 中找 **两个互不重叠的子数组** 且它们的和都等于 target 。可能会有多种方案，请你返回满足要求的两个子数组长度和的 **最小值** 。

请返回满足要求的最小长度和，如果无法找到这样的两个子数组，请返回 **-1** 。

示例 1：

```
输入：arr = [3,2,2,4,3], target = 3
输出：2
解释：只有两个子数组和为 3 （[3] 和 [3]）。它们的长度和为 2 。
```

示例 2：

```
输入：arr = [7,3,4,7], target = 7
输出：2
解释：尽管我们有 3 个互不重叠的子数组和为 7 （[7], [3,4] 和 [7]），但我们会选择第一个和第三个子数组，因为它们的长度和 2 是最小值。
```

示例 3：

```
输入：arr = [4,3,2,6,2,3,4], target = 6
输出：-1
解释：我们只有一个和为 6 的子数组。
```

示例 4：

```
输入：arr = [5,5,4,4,5], target = 3
输出：-1
解释：我们无法找到和为 3 的子数组。
```

示例 5：

```
输入：arr = [3,1,1,1,5,1,2,1], target = 3
输出：3
解释：注意子数组 [1,2] 和 [2,1] 不能成为一个方案因为它们重叠了。
```

提示：

- 1 <= arr.length <= 10^5
- 1 <= arr[i] <= 1000
- 1 <= target <= 10^8

双指针。

首先用双指针抠出所有的可行区间。

把这些可行区间按左端点排序，右端点小的靠前。

从后往前扫一遍，利用map可以找到左端点大于当前点的右端点的第一个区间下标是多少。设这个下标为p[i]

然后做一个st表，存区间长度的最小值。

对每个区间i，更新答案为自己的长度加上[p[i],n]区间中st表维护的最短长度。

1470. 重新排列数组

给你一个数组 nums ，数组中有 2n 个元素，按 $[x_1,x_2,...,x_n,y_1,y_2,...,y_n]$ 的格式排列。

请你将数组按 $[x_1,y_1,x_2,y_2,...,x_n,y_n]$ 格式重新排列，返回重排后的数组。

示例 1：

```
输入：nums = [2,5,1,3,4,7], n = 3
输出：[2,3,5,4,1,7]
解释：由于 x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 ，所以答案为 [2,3,5,4,1,7]
```

示例 2：

```
输入：nums = [1,2,3,4,4,3,2,1], n = 4
输出：[1,4,2,3,3,2,4,1]
```

示例 3：

```
输入：nums = [1,1,2,2], n = 2
输出：[1,2,1,2]
```

提示：

- 1 <= n <= 500
- nums.length == 2n
- 1 <= nums[i] <= 10^3

水题。

两个指针维护一下位置即可。

1471. 数组中的 k 个最强值

给你一个整数数组 arr 和一个整数 k 。

设 m 为数组的中位数，只要满足下述两个前提之一，就可以判定 arr[i] 的值比 arr[j] 的值更强：

- |arr[i] - m| > |arr[j] - m|
- |arr[i] - m| == |arr[j] - m| ，且 arr[i] > arr[j]

请返回由数组中最强的 k 个值组成的列表。答案可以以 **任意顺序** 返回。

中位数 是一个有序整数列表中处于中间位置的值。形式上，如果列表的长度为 n ，那么中位数就是该有序列表（下标从 0 开始）中位于 ((n - 1) / 2) 的元素。

- 例如 arr = [6, -3, 7, 2, 11] ， n = 5 ：数组排序后得到 arr = [-3, 2, 6, 7, 11] ，数组的中间位置为 m = ((5 - 1) / 2) = 2 ，中位数 arr[m] 的值为 6 。
- 例如 arr = [-7, 22, 17, 3] ， n = 4 ：数组排序后得到 arr = [-7, 3, 17, 22] ，数组的中间位置为 m = ((4 - 1) / 2) = 1 ，中位数 arr[m] 的值为 3 。

示例 1：

```
输入：arr = [1,2,3,4,5], k = 2
输出：[5,1]
解释：中位数为 3，按从强到弱顺序排序后，数组变为 [5,1,4,2,3]。最强的两个元素是 [5, 1]。[1, 5] 也是正确答案。
注意，尽管 |5 - 3| == |1 - 3| ，但是 5 比 1 更强，因为 5 > 1 。
```

示例 2：

```
输入：arr = [1,1,3,5,5], k = 2
输出：[5,5]
解释：中位数为 3，按从强到弱顺序排序后，数组变为 [5,5,1,1,3]。最强的两个元素是 [5, 5]。
```

示例 3：

```
输入：arr = [6,7,11,7,6,8], k = 5
输出：[11,8,6,6,7]
解释：中位数为 7，按从强到弱顺序排序后，数组变为 [11,8,6,6,7,7]。
[11,8,6,6,7] 的任何排列都是正确答案。
```

示例 4：

```
输入：arr = [6,-3,7,2,11], k = 3
输出：[-3,11,2]
```

示例 5：

```
输入：arr = [-7,22,17,3], k = 2
输出：[22,17]
```

提示：

- 1 <= arr.length <= 10^5
- -10^5 <= arr[i] <= 10^5
- 1 <= k <= arr.length

水题。

找到中位数，对每个数字i塞一个pair：(abs(i-m),i)进去，再排序。取最大的k个即可。

1472. 设计浏览器历史记录

你有一个只支持单个标签页的 **浏览器** ，最开始你浏览的网页是 homepage ，你可以访问其他的网站 url ，也可以在浏览历史中后退 steps 步或前进 steps 步。

请你实现 BrowserHistory 类：

- BrowserHistory(string homepage) ，用 homepage 初始化浏览器类。

- `void visit(string url)` 从当前页跳转访问 `url` 对应的页面 。执行此操作会把浏览历史前进的记录全部删除。
- `string back(int steps)` 在浏览历史中后退 `steps` 步。如果你只能在浏览历史中后退至多 `x` 步且 `steps > x` ，那么你只后退 `x` 步。请返回后退 **至多** `steps` 步以后的 `url` 。
- `string forward(int steps)` 在浏览历史中前进 `steps` 步。如果你只能在浏览历史中前进至多 `x` 步且 `steps > x` ，那么你只前进 `x` 步。请返回前进 **至多** `steps` 步以后的 `url` 。

示例:

```
输入：
["BrowserHistory","visit","visit","visit","back","back","forward","visit","forward","back","back"]
[["leetcode.com"],["google.com"],["facebook.com"],["youtube.com"],[1],[1],[1],["linkedin.com"],[2],[2],[7]]
输出：
[null,null,null,null,"facebook.com","google.com","facebook.com",null,"linkedin.com","google.com","leetcode.com"]

解释：
BrowserHistory browserHistory = new BrowserHistory("leetcode.com");
browserHistory.visit("google.com");       // 你原本在浏览 "leetcode.com" 。访问 "google.com"
browserHistory.visit("facebook.com");     // 你原本在浏览 "google.com" 。访问 "facebook.com"
browserHistory.visit("youtube.com");      // 你原本在浏览 "facebook.com" 。访问 "youtube.com"
browserHistory.back(1);                   // 你原本在浏览 "youtube.com" ，后退到 "facebook.com" 并返回 "facebook.com"
browserHistory.back(1);                   // 你原本在浏览 "facebook.com" ，后退到 "google.com" 并返回 "google.com"
browserHistory.forward(1);                // 你原本在浏览 "google.com" ，前进到 "facebook.com" 并返回 "facebook.com"
browserHistory.visit("linkedin.com");     // 你原本在浏览 "facebook.com" 。 访问 "linkedin.com"
browserHistory.forward(2);                // 你原本在浏览 "linkedin.com" ，你无法前进任何步数。
browserHistory.back(2);                   // 你原本在浏览 "linkedin.com" ，后退两步依次先到 "facebook.com" ，然后到 "google
browserHistory.back(7);                   // 你原本在浏览 "google.com"， 你只能后退一步到 "leetcode.com" ，并返回 "leetcode
```

提示:

- `1 <= homepage.length <= 20`
- `1 <= url.length <= 20`
- `1 <= steps <= 100`
- `homepage` 和 `url` 都只包含 '.' 或者小写英文字母。
- 最多调用 `5000` 次 `visit`， `back` 和 `forward` 函数。

双向链表

注意用lst维护链表尾在哪儿，实际上也就是当前在哪个页面。

但是如果没有访问新的页面，注意forward操作还是可以访问到lst的nxt节点的。

初始节点插入一个homepage即可。

1473. 给房子涂色 III

在一个小城市里，有 `m` 个房子排成一排，你需要给每个房子涂上 `n` 种颜色之一（颜色编号为 `1` 到 `n` ）。有的房子去年夏天已经涂过颜色了，所以这些房子不需要被重新涂色。

我们将连续相同颜色尽可能多的房子称为一个街区。（比方说 `houses = [1,2,2,3,3,2,1,1]` ，它包含 5 个街区 `[{1}, {2,2}, {3,3}, {2}, {1,1}]` 。）

给你一个数组 `houses` ，一个 `m * n` 的矩阵 `cost` 和一个整数 `target` ，其中：

- `houses[i]`：是第 `i` 个房子的颜色，**0** 表示这个房子还没有被涂色。
- `cost[i][j]`：是将第 `i` 个房子涂成颜色 `j+1` 的花费。

请你返回房子涂色方案的最小总花费，使得每个房子都被涂色后，恰好组成 `target` 个街区。如果没有可用的涂色方案，请返回 **-1** 。

示例 1:

```
输入：houses = [0,0,0,0,0], cost = [[1,10],[10,1],[10,1],[1,10],[5,1]], m = 5, n = 2, target = 3
输出：9
解释：房子涂色方案为 [1,2,2,1,1]
此方案包含 target = 3 个街区，分别是 [{1}, {2,2}, {1,1}]。
涂色的总花费为 (1 + 1 + 1 + 1 + 5) = 9。
```

示例 2：

```
输入：houses = [0,2,1,2,0], cost = [[1,10],[10,1],[10,1],[1,10],[5,1]], m = 5, n = 2, target = 3
输出：11
解释：有的房子已经被涂色了，在此基础上涂色方案为 [2,2,1,2,2]
此方案包含 target = 3 个街区，分别是 [{2,2}, {1}, {2,2}]。
给第一个和最后一个房子涂色的花费为 (10 + 1) = 11。
```

示例 3：

```
输入：houses = [0,0,0,0,0], cost = [[1,10],[10,1],[1,10],[10,1],[1,10]], m = 5, n = 2, target = 5
输出：5
```

示例 4：

```
输入：houses = [3,1,2,3], cost = [[1,1,1],[1,1,1],[1,1,1],[1,1,1]], m = 4, n = 3, target = 3
输出：-1
解释：房子已经被涂色并组成了 4 个街区，分别是 [{3},{1},{2},{3}] ，无法形成 target = 3 个街区。
```

提示：

- `m == houses.length == cost.length`
- `n == cost[i].length`
- `1 <= m <= 100`
- `1 <= n <= 20`
- `1 <= target <= m`
- `0 <= houses[i] <= n`
- `1 <= cost[i][j] <= 10^4`

---

dp。方程：

颜色相同：dp[i][j][k]=min(dp[i][j][k],dp[i-1][j][k]+cost[i-1][j-1]);

颜色不同：dp[i][j][k]=min(dp[i][j][k],dp[i-1][l][k-1]+cost[i-1][j-1]);

dp[i][j][k]表示第i个位置，涂成j颜色，目前已经有k段颜色。

那些涂过颜色的点不用考虑cost。

1491. 去掉最低工资和最高工资后的工资平均值

给你一个整数数组 `salary` ，数组里每个数都是 **唯一** 的，其中 `salary[i]` 是第 `i` 个员工的工资。

请你返回去掉最低工资和最高工资以后，剩下员工工资的平均值。

示例 1：

```
输入：salary = [4000,3000,1000,2000]
输出：2500.00000
解释：最低工资和最高工资分别是 1000 和 4000 。
去掉最低工资和最高工资以后的平均工资是 (2000+3000)/2= 2500
```

示例 2：

```
输入：salary = [1000,2000,3000]
输出：2000.00000
解释：最低工资和最高工资分别是 1000 和 3000 。
去掉最低工资和最高工资以后的平均工资是 (2000)/1= 2000
```

示例 3：

```
输入：salary = [6000,5000,4000,3000,2000,1000]
输出：3500.00000
```

示例 4：

```
输入：salary = [8000,9000,2000,3000,6000,1000]
输出：4750.00000
```

提示：

- `3 <= salary.length <= 100`
- `10^3 <= salary[i] <= 10^6`
- `salary[i]` 是唯一的。
- 与真实值误差在 `10^-5` 以内的结果都将视为正确答案。

水题，按题意来。

记得算平均数的时候减个二。

1492. n 的第 k 个因子

给你两个正整数 `n` 和 `k` 。

如果正整数 `i` 满足 `n % i == 0` ，那么我们就说正整数 `i` 是整数 `n` 的因子。

考虑整数 `n` 的所有因子，将它们 **升序排列** 。请你返回第 `k` 个因子。如果 `n` 的因子数少于 `k` ，请你返回 **-1** 。

示例 1：

```
输入：n = 12, k = 3
输出：3
解释：因子列表包括 [1, 2, 3, 4, 6, 12]，第 3 个因子是 3 。
```

示例 2：

```
输入：n = 7, k = 2
输出：7
解释：因子列表包括 [1, 7] ，第 2 个因子是 7 。
```

示例 3：

```
输入：n = 4, k = 4
输出：-1
解释：因子列表包括 [1, 2, 4] ，只有 3 个因子，所以我们应该返回 -1 。
```

示例 4：

```
输入：n = 1, k = 1
输出：1
解释：因子列表包括 [1] ，第 1 个因子为 1 。
```

示例 5：

```
输入：n = 1000, k = 3
输出：4
解释：因子列表包括 [1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000] 。
```

提示：

- `1 <= k <= n <= 1000`

水题，求因子数。

把因子全部找出来就完事了。

1493. 删掉一个元素以后全为 1 的最长子数组

给你一个二进制数组 `nums` ，你需要从中删掉一个元素。

请你在删掉元素的结果数组中，返回最长的且只包含 1 的非空子数组的长度。

如果不存在这样的子数组，请返回 0 。

提示 1：

```
输入：nums = [1,1,0,1]
输出：3
解释：删掉位置 2 的数后，[1,1,1] 包含 3 个 1 。
```

示例 2：

```
输入：nums = [0,1,1,1,0,1,1,0,1]
输出：5
解释：删掉位置 4 的数字后，[0,1,1,1,1,1,0,1] 的最长全 1 子数组为 [1,1,1,1,1] 。
```

示例 3：

```
输入：nums = [1,1,1]
输出：2
解释：你必须要删除一个元素。
```

示例 4：

```
输入：nums = [1,1,0,0,1,1,1,0,1]
输出：4
```

示例 5：

```
输入：nums = [0,0,0]
输出：0
```

提示：

- `1 <= nums.length <= 10^5`
- `nums[i]` 要么是 `0` 要么是 `1` 。

相邻相同数字的压缩。

把1和0压到一起，可以用pair存储。

首先用每段1的长度更新答案，然后对单个0计算它两侧的1加在一起的长度更新答案。

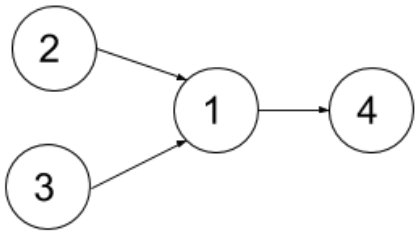
1494. 并行课程 II

给你一个整数 n 表示某所大学里课程的数目，编号为 1 到 n ，数组 dependencies 中， dependencies[i] = [x_i, y_i] 表示一个先修课的关系，也就是课程 x_i 必须在课程 y_i 之前上。同时你还有一个整数 k 。

在一个学期中，你 **最多** 可以同时上 k 门课，前提是这些课的先修课在之前的学期里已经上过了。

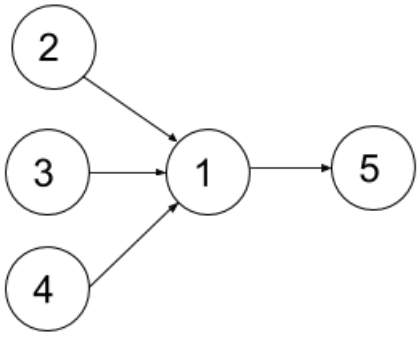
请你返回上完所有课最少需要多少个学期。题目保证一定存在一种上完所有课的方式。

示例 1：




```
输入：n = 4, dependencies = [[2,1],[3,1],[1,4]], k = 2
输出：3
解释：上图展示了题目输入的图。在第一个学期中，我们可以上课程 2 和课程 3 。然后第二个学期上课程 1 ，第三个学期上课程 4 。
```

示例 2：




```
输入：n = 5, dependencies = [[2,1],[3,1],[4,1],[1,5]], k = 2
输出：4
解释：上图展示了题目输入的图。一个最优方案是：第一学期上课程 2 和 3，第二学期上课程 4 ，第三学期上课程 1 ，第四学期上课程 5 。
```

示例 3：

```
输入：n = 11, dependencies = [], k = 2
输出：6
```

提示：

- 1 <= n <= 15
- 1 <= k <= n
- 0 <= dependencies.length <= n * (n-1) / 2
- dependencies[i].length == 2
- 1 <= x_i, y_i <= n
- x_i != y_i
- 所有先修关系都是不同的，也就是说 dependencies[i] != dependencies[j] 。
- 题目输入的图是个有向无环图。

枚举子集和，(s-1)&msk，复杂度O(3^n)。

首先对每个点处理出它的前置节点，为了方便后续处理，可以使用单个数字的不同位来进行存储，代码如下：

```
            pre[i[1]]|=(1<<i[0]);
```

接下来就是枚举子集和，最外层循环是传统的O(2^n)枚举。

内层先要处理出从当前状态可以抵达哪些节点，代码如下：

```
            for(int i=0;i<n;++i)
                if((pre[i]&msk)==pre[i])
                    nxt|=(1<<i);
```

然后要去除掉msk当中已经有的节点，枚举添加新的节点，代码如下：

```
            nxt&=~msk;
```

最后一步就是枚举新加节点的子集即可，代码如下：

```
            for(int s=nxt;s;s=(s-1)&nxt)
                if(__builtin_popcount(s)<=k)
                    dp[msk|s]=min(dp[msk|s],dp[msk]+1);
```

不难发现本质是一个用前面的状态更新后面的值的dp。

1480. 一维数组的动态和

给你一个数组 `nums` 。数组「动态和」的计算公式为： `runningSum[i] = sum(nums[0]…nums[i])` 。

请返回 `nums` 的动态和。

示例 1：

```
输入：nums = [1,2,3,4]
输出：[1,3,6,10]
解释：动态和计算过程为 [1, 1+2, 1+2+3, 1+2+3+4] 。
```

示例 2：

```
输入：nums = [1,1,1,1,1]
输出：[1,2,3,4,5]
解释：动态和计算过程为 [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1] 。
```

示例 3：

```
输入：nums = [3,1,2,10,1]
输出：[3,4,6,16,17]
```

提示：

- `1 <= nums.length <= 1000`
- `-10^6 <= nums[i] <= 10^6`

水题。

求前缀和。

1481. 不同整数的最少数目

给你一个整数数组 `arr` 和一个整数 `k` 。现需要从数组中恰好移除 `k` 个元素，请找出移除后数组中不同整数的最少数目。

示例 1：

```
输入：arr = [5,5,4], k = 1
输出：1
解释：移除 1 个 4 ，数组中只剩下 5 一种整数。
```

示例 2：

```
输入：arr = [4,3,1,1,3,3,2], k = 3
输出：2
解释：先移除 4、2 ，然后再移除两个 1 中的任意 1 个或者三个 3 中的任意 1 个，最后剩下 1 和 3 两种整数。
```

提示：

- 1 <= arr.length <= 10^5
- 1 <= arr[i] <= 10^9
- 0 <= k <= arr.length

水题。

找到每个数字的出现次数，按照从小到大排序，贪心地删除出现次数最少的即可。

1482. 制作 m 束花所需的最少天数

给你一个整数数组 bloomDay ，以及两个整数 m 和 k 。

现需要制作 m 束花。制作花束时，需要使用花园中 **相邻的 k 朵花** 。

花园中有 n 朵花，第 i 朵花会在 bloomDay[i] 时盛开，**恰好** 可以用于 **一束** 花中。

请你返回从花园中摘 m 束花需要等待的最少的天数。如果不能摘到 m 束花则返回 **-1** 。

示例 1：

```
输入：bloomDay = [1,10,3,10,2], m = 3, k = 1
输出：3
解释：让我们一起观察这三天的花开过程，x 表示花开，而 _ 表示花还未开。
现在需要制作 3 束花，每束只需要 1 朵。
1 天后：[x, _, _, _, _]   // 只能制作 1 束花
2 天后：[x, _, _, _, x]   // 只能制作 2 束花
3 天后：[x, _, x, _, x]   // 可以制作 3 束花，答案为 3
```

示例 2：

```
输入：bloomDay = [1,10,3,10,2], m = 3, k = 2
输出：-1
解释：要制作 3 束花，每束需要 2 朵花，也就是一共需要 6 朵花。而花园中只有 5 朵花，无法满足制作要求，返回 -1 。
```

示例 3：

```
输入：bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3
输出：12
解释：要制作 2 束花，每束需要 3 朵。
花园在 7 天后和 12 天后的情况如下：
7 天后：[x, x, x, x, _, x, x]
可以用前 3 朵盛开的花制作第一束花。但不能使用后 3 朵盛开的花，因为它们不相邻。
12 天后：[x, x, x, x, x, x, x]
显然，我们可以用不同的方式制作两束花。
```

示例 4：

```
输入：bloomDay = [1000000000,1000000000], m = 1, k = 1
输出：1000000000
解释：需要等 1000000000 天才能采到花来制作花束
```

示例 5：

```
输入：bloomDay = [1,10,2,9,3,8,4,7,5,6], m = 4, k = 2
输出：9
```

提示：

- bloomDay.length == n
- 1 <= n <= 10^5
- 1 <= bloomDay[i] <= 10^9
- 1 <= m <= 10^6
- 1 <= k <= n

二分

二分天数，O(n)来check。

check的时候我是抠出了所有连续段，然后把连续段长度都除以k，然后与m比较大小。

1486. 数组异或操作

给你两个整数，n 和 start 。

数组 nums 定义为：nums[i] = start + 2*i （下标从 0 开始）且 n == nums.length 。

请返回 nums 中所有元素按位异或（**XOR**）后得到的结果。

示例 1：

```
输入：n = 5, start = 0
输出：8
解释：数组 nums 为 [0, 2, 4, 6, 8]，其中 (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8 。
     "^" 为按位异或 XOR 运算符。
```

示例 2：

```
输入：n = 4, start = 3
输出：8
解释：数组 nums 为 [3, 5, 7, 9]，其中 (3 ^ 5 ^ 7 ^ 9) = 8.
```

示例 3：

```
输入：n = 1, start = 7
输出：7
```

示例 4：

```
输入：n = 10, start = 5
输出：2
```

提示：

- 1 <= n <= 1000
- 0 <= start <= 1000
- n == nums.length

水题。

算总异或就行。

1487. 保证文件名唯一

给你一个长度为 `n` 的字符串数组 `names` 。你将会在文件系统中创建 `n` 个文件夹：在第 `i` 分钟，新建名为 `names[i]` 的文件夹。

由于两个文件 **不能** 共享相同的文件名，因此如果新建文件夹使用的文件名已经被占用，系统会以 `(k)` 的形式为新文件夹的文件名添加后缀，其中 `k` 是能保证文件名唯一的 **最小正整数** 。

返回长度为 *`n`* 的字符串数组，其中 `ans[i]` 是创建第 `i` 个文件夹时系统分配给该文件夹的实际名称。

示例 1：

```
输入：names = ["pes","fifa","gta","pes(2019)"]
输出：["pes","fifa","gta","pes(2019)"]
解释：文件系统将会这样创建文件名：
"pes" --> 之前未分配，仍为 "pes"
"fifa" --> 之前未分配，仍为 "fifa"
"gta" --> 之前未分配，仍为 "gta"
"pes(2019)" --> 之前未分配，仍为 "pes(2019)"
```

示例 2：

```
输入：names = ["gta","gta(1)","gta","avalon"]
输出：["gta","gta(1)","gta(2)","avalon"]
解释：文件系统将会这样创建文件名：
"gta" --> 之前未分配，仍为 "gta"
"gta(1)" --> 之前未分配，仍为 "gta(1)"
"gta" --> 文件名被占用，系统为该名称添加后缀 (k)，由于 "gta(1)" 也被占用，所以 k = 2 。实际创建的文件名为 "gta(2)" 。
"avalon" --> 之前未分配，仍为 "avalon"
```

示例 3：

```
输入：names = ["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece"]
输出：["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece(4)"]
解释：当创建最后一个文件夹时，最小的正有效 k 为 4 ，文件名变为 "onepiece(4)"。
```

示例 4：

```
输入：names = ["wano","wano","wano","wano"]
输出：["wano","wano(1)","wano(2)","wano(3)"]
解释：每次创建文件夹 "wano" 时，只需增加后缀中 k 的值即可。
```

示例 5：

```
输入：names = ["kaido","kaido(1)","kaido","kaido(1)"]
输出：["kaido","kaido(1)","kaido(2)","kaido(1)(1)"]
解释：注意，如果含后缀文件名被占用，那么系统也会按规则在名称后添加新的后缀 (k) 。
```

提示：

- `1 <= names.length <= 5 * 10^4`
- `1 <= names[i].length <= 20`
- `names[i]` 由小写英文字母、数字和/或圆括号组成。

---

map教学。

用map记录出现过的单词，以及此时应该使用的编号。

如果某个编号已经被使用过，就不断往下找直到找到第一个没有被使用的编号。

1488. 避免洪水泛滥

你的国家有无数个湖泊，所有湖泊一开始都是空的。当第 `n` 个湖泊下雨的时候，如果第 `n` 个湖泊是空的，那么它就会装满水，否则这个湖泊会发生洪水。你的目标是避免任意一个湖泊发生洪水。

给你一个整数数组 `rains` ，其中：

- `rains[i] > 0` 表示第 `i` 天时，第 `rains[i]` 个湖泊会下雨。
- `rains[i] == 0` 表示第 `i` 天没有湖泊会下雨，你可以选择 **一个** 湖泊并 **抽干** 这个湖泊的水。

请返回一个数组 `ans` ，满足：

- `ans.length == rains.length`
- 如果 `rains[i] > 0` ，那么 `ans[i] == -1` 。
- 如果 `rains[i] == 0` ， `ans[i]` 是你第 `i` 天选择抽干的湖泊。

如果有多种可行解，请返回它们中的 **任意一个** 。如果没办法阻止洪水，请返回一个 **空的数组** 。

请注意，如果你选择抽干一个装满水的湖泊，它会变成一个空的湖泊。但如果你选择抽干一个空的湖泊，那么将无事发生（详情请看示例 4）。

示例 1：

```
输入：rains = [1,2,3,4]
输出：[-1,-1,-1,-1]
解释：第一天后，装满水的湖泊包括 [1]
第二天后，装满水的湖泊包括 [1,2]
第三天后，装满水的湖泊包括 [1,2,3]
第四天后，装满水的湖泊包括 [1,2,3,4]
没有哪一天你可以抽干任何湖泊的水，也没有湖泊会发生洪水。
```

示例 2：

```
输入：rains = [1,2,0,0,2,1]
输出：[-1,-1,2,1,-1,-1]
解释：第一天后，装满水的湖泊包括 [1]
第二天后，装满水的湖泊包括 [1,2]
第三天后，我们抽干湖泊 2 。所以剩下装满水的湖泊包括 [1]
第四天后，我们抽干湖泊 1 。所以暂时没有装满水的湖泊了。
第五天后，装满水的湖泊包括 [2]。
第六天后，装满水的湖泊包括 [1,2]。
可以看出，这个方案下不会有洪水发生。同时， [-1,-1,1,2,-1,-1] 也是另一个可行的没有洪水的方案。
```

示例 3：

```
输入：rains = [1,2,0,1,2]
输出：[]
解释：第二天后，装满水的湖泊包括 [1,2]。我们可以在第三天抽干一个湖泊的水。
但第三天后，湖泊 1 和 2 都会再次下雨，所以不管我们第三天抽干哪个湖泊的水，另一个湖泊都会发生洪水。
```

示例 4：

```
输入：rains = [69,0,0,0,69]
输出：[-1,69,1,1,-1]
解释：任何形如 [-1,69,x,y,-1], [-1,x,69,y,-1] 或者 [-1,x,y,69,-1] 都是可行的解，其中 1 <= x,y <= 10^9
```

示例 5：

```
输入：rains = [10,20,20]
输出：[]
解释：由于湖泊 20 会连续下 2 天的雨，所以没有没有办法阻止洪水。
```

提示：

- `1 <= rains.length <= 10^5`
- `0 <= rains[i] <= 10^9`

---

set教学。

用set记录可以放水的位置。

对每个需要放水的位置lower_bound一下它上一次出现和这一次出现中有没有0的位置。

上一次出现可以用map记录。

如果没有，直接返回false；如果有，set中删除对应编号。

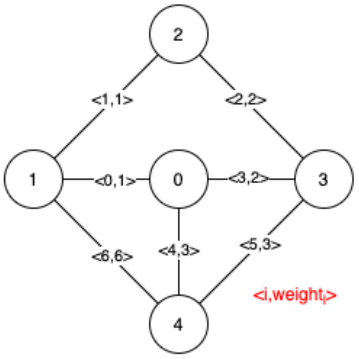
1489. 找到最小生成树里的关键边和伪关键边

给你一个 `n` 个点的带权无向连通图，节点编号为 `0` 到 `n-1` ，同时还有一个数组 `edges` ，其中 `edges[i] = [from`$_i$`, to`$_i$`, weight`$_i$`]` 表示在 `from`$_i$ 和 `to`$_i$ 节点之间有一条带权无向边。最小生成树 (MST) 是给定图中边的一个子集，它连接了所有节点且没有环，而且这些边的权值和最小。

请你找到给定图中最小生成树的所有关键边和伪关键边。如果从图中删去某条边，会导致最小生成树的权值和增加，那么我们就说它是一条关键边。伪关键边则是可能会出现在某些最小生成树中但不会出现在所有最小生成树中的边。

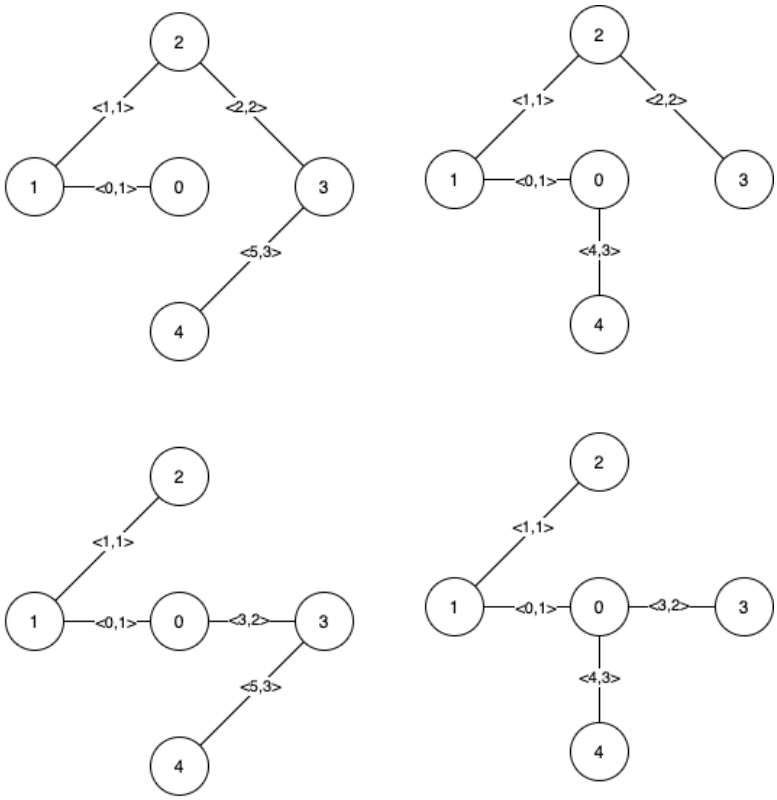
请注意，你可以分别以任意顺序返回关键边的下标和伪关键边的下标。

示例 1：



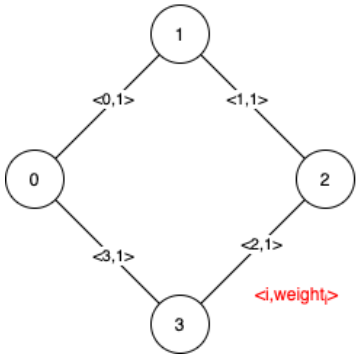
```
输入：n = 5, edges = [[0,1,1],[1,2,1],[2,3,2],[0,3,2],[0,4,3],[3,4,3],[1,4,6]]
输出：[[0,1],[2,3,4,5]]
解释：上图描述了给定图。
```

下图是所有的最小生成树。



注意到第 `0` 条边和第 `1` 条边出现在了所有最小生成树中，所以它们是关键边，我们将这两个下标作为输出的第一个列表。
边 `2`，`3`，`4` 和 `5` 是所有 `MST` 的剩余边，所以它们是伪关键边。我们将它们作为输出的第二个列表。

示例 2：




```
输入：n = 4, edges = [[0,1,1],[1,2,1],[2,3,1],[0,3,1]]
输出：[[],[0,1,2,3]]
解释：可以观察到 4 条边都有相同的权值，任选它们中的 3 条可以形成一棵 MST 。所以 4 条边都是伪关键边。
```

提示：

- `2 <= n <= 100`
- `1 <= edges.length <= min(200, n * (n - 1) / 2)`
- `edges[i].length == 3`
- $0 <= from_i < to_i < n$
- $1 <= weight_i <= 1000$
- 所有 $(from_i, to_i)$ 数对都是互不相同的。

最小生成树结论。

范围很小，可以用枚举边的方法搞过。

注意struct比pair快，能一次排序解决就不要优先队列。

关键边的性质其实是去掉这条边之后最小生成树权值和变大了。

伪关键边的性质其实是把这条边加入生成树之后再跑最小生成树，权值和不变。

利用kruskal可以很方便的完成上述两种判断：加入生成树的操作等价于提前在并查集中合并这条边两端的节点。

注意用kk时判断全图是否连通，不连通权值和为正无穷。

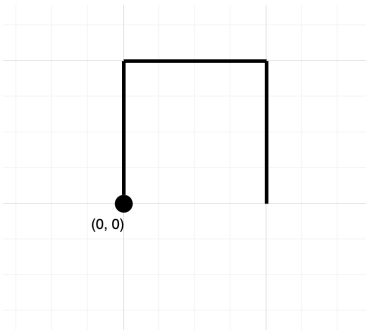
1496. 判断路径是否相交

给你一个字符串 `path`，其中 `path[i]` 的值可以是 `'N'`、`'S'`、`'E'` 或者 `'W'`，分别表示向北、向南、向东、向西移动一个单位。

机器人从二维平面上的原点 `(0, 0)` 处开始出发，按 `path` 所指示的路径行走。

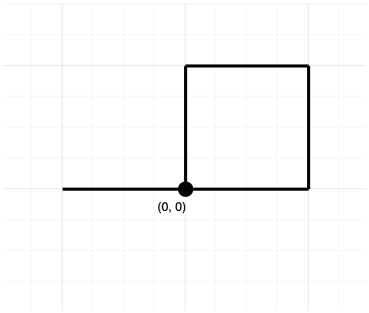
如果路径在任何位置上出现相交的情况，也就是走到之前已经走过的位置，请返回 `True` ；否则，返回 `False` 。

示例 1：




```
输入：path = "NES"
输出：false
解释：该路径没有在任何位置相交。
```

示例 2：




```
输入：path = "NESWW"
输出：true
解释：该路径经过原点两次。
```

提示：

- `1 <= path.length <= 10^4`
- `path` 仅由 `{'N', 'S', 'E', 'W}` 中的字符组成

水题。map加pair教程。

注意判断一下终点是否已经经过一次，不然会WA。

1497. 检查数组对是否可以被 k 整除

给你一个整数数组 `arr` 和一个整数 `k` ，其中数组长度是偶数，值为 `n` 。

现在需要把数组恰好分成 `n / 2` 对，以使每对数字的和都能够被 `k` 整除。

如果存在这样的分法，请返回 *True* ；否则，返回 *False* 。

示例 1：

```
输入：arr = [1,2,3,4,5,10,6,7,8,9], k = 5
输出：true
解释：划分后的数字对为 (1,9),(2,8),(3,7),(4,6) 以及 (5,10) 。
```

示例 2：

```
输入：arr = [1,2,3,4,5,6], k = 7
输出：true
解释：划分后的数字对为 (1,6),(2,5) 以及 (3,4) 。
```

示例 3：

```
输入：arr = [1,2,3,4,5,6], k = 10
输出：false
解释：无法在将数组中的数字分为三对的同时满足每对数字和能够被 10 整除的条件。
```

示例 4：

```
输入：arr = [-10,10], k = 2
输出：true
```

示例 5：

```
输入：arr = [-1,1,-2,2,-3,3,-4,4], k = 3
输出：true
```

提示：

- `arr.length == n`
- `1 <= n <= 10^5`
- `n` 为偶数
- `-10^9 <= arr[i] <= 10^9`
- `1 <= k <= 10^5`

---

map教学。

每个数字对k取个余，保存每个数字的出现次数。

由于有负数的存在，所以要考虑下面三种情况：

1. i+(k-i)
2. i+(-k-i)
3. i+(-i)

此外还要考虑以上三种情况相等时数字的出现个数问题。

此外贪心的策略简单证明一下就会发现是正确的，能找到一对就减去一对，最后比较一下有没有取完即可。

LCP 01. 猜数字

小A 和 小B 在玩猜数字。小B 每次从 1, 2, 3 中随机选择一个，小A 每次也从 1, 2, 3 中选择一个猜。他们一共进行三次这个游戏，请返回 小A 猜对了几次?

输入的 guess 数组为 小A 每次的猜测， answer 数组为 小B 每次的选择。 guess 和 answer 的长度都等于3。

示例 1：

```
输入：guess = [1,2,3], answer = [1,2,3]
输出：3
解释：小A 每次都猜对了。
```

示例 2：

```
输入：guess = [2,2,3], answer = [3,2,1]
输出：1
解释：小A 只猜对了第二次。
```

限制：

1. guess 的长度 = 3
2. answer 的长度 = 3
3. guess 的元素取值为 {1, 2, 3} 之一。
4. answer 的元素取值为 {1, 2, 3} 之一。

水题，比较数字是否相等。

面试题 02.01. 移除重复节点

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

示例1:

```
输入：[1, 2, 3, 3, 2, 1]
输出：[1, 2, 3]
```

示例2:

```
输入：[1, 1, 1, 1, 2]
输出：[1, 2]
```

提示：

1. 链表长度在[0, 20000]范围内。
2. 链表元素在[0, 20000]范围内。

进阶：

如果不得使用临时缓冲区，该怎么解决?

水题。

链表操作，删除重复出现的值节点，保留第一个出现的节点。

搞两个指针p、q，q不断向后找，如果非重复就让p的next指向q，并且把p向后移动一位。

利用map记录出现的数字。

具体参考代码：

```
        while(p!=NULL&&q!=NULL) {
            if(mp[q->val]) {
                q=q->next;
                continue;
            }
            mp[q->val]=1;
            p->next=q;
            p=p->next;
            q=q->next;
        }
        p->next=q;
```

注意上面代码已经把头节点加入了map，所以要判断一下head是否为空。

剑指 Offer 09. 用两个栈实现队列

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

水题。利用辅助栈颠倒栈的顺序。

入队的操作直接向A栈压入数据。

出队的操作首先进行判断，如果A栈没有数据则返回-1。

如果有数据，把A栈中的所有元素依次序放入B栈，相当于颠倒了A栈顺序，此时B栈栈顶就是答案。

取出B栈栈顶后再把B栈中的所有元素放入A栈即可。

剑指 Offer 29. 顺时针打印矩阵

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

示例 1：

```
输入：matrix = [[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：matrix = [[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]
```

限制：

- `0 <= matrix.length <= 100`
- `0 <= matrix[i].length <= 100`

注意：本题与主站 54 题相同：https://leetcode-cn.com/problems/spiral-matrix/ (https://leetcode-cn.com/problems/spiral-matrix/)

细节大模拟

主要考虑过程是下面这几步：

1. 用 vis 数组记录某个位置是否被访问过，访问过的话不再访问。同时用 cnt 记录当前已经访问了多少个数字，当 cnt == n * m 时跳出循环。
2. 模拟四个方向，在每个方向上走到不能走为止，把所有数字逐个添加进答案。
3. 转换方向的时候，**首先要把出界的下标拉回来，然后向着下一个方向前进一步，因为当前位置已经被访问过了。**

剑指 Offer 22. 链表中倒数第k个节点

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。例如，一个链表有6个节点，从头节点开始，它们的值依次是1、2、3、4、5、6。这个链表的倒数第3个节点是值为4的节点。

示例：

```
给定一个链表：1->2->3->4->5, 和 k = 2.

返回链表 4->5.
```

---

dfs

正着扫一遍获得总节点数 tot ， 再正着扫一遍找正数的第 tot - k + 1 个节点即可。

剑指 Offer 46. 把数字翻译成字符串

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`

DP

分解数字类的DP都是这种套路：

```
            dp[i]=dp[i-1];
            if(n[i-2]==1) dp[i]+=dp[i-2];
            if(n[i-2]==2&&n[i-1]>=0&&n[i-1]<=5) dp[i]+=dp[i-2];
```

累加一下前面的情况即可。

剑指 Offer 64. 求1+2+...+n

求 1+2+...+n ，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

示例 1：

```
输入：n = 3
输出：6
```

示例 2：

```
输入：n = 9
输出：45
```

限制：

- 1 <= n <= 10000

思维题

不用循环和判断来算自然数列前n项和。基本方案有两种：

1. 用短路来做dfs的判断，代码如下：

```
int dfs(int x) {
     x && (x+=dfs(x-1));
     return x;
 }
```

2. 搞一个快速乘，大概长下面这样：

```
int ans=0,a=n,b=n+1;
for(int i=0;i<14;++i) {
     if(b&1) ans+=a;
     b>>=1;
     a<<=1;
}
return ans/2;
```

仍然把 if 用短路代替掉，之后把循环体复制粘贴14遍即可。

面试题 16.18. 模式匹配

你有两个字符串，即 pattern 和 value 。 pattern 字符串由字母 "a" 和 "b" 组成，用于描述字符串中的模式。例如，字符串 "catcatgocatgo" 匹配模式 "aabab" （其中 "cat" 是 "a"， "go" 是 "b" ），该字符串也匹配像 "a" 、 "ab" 和 "b" 这样的模式。但需注意 "a" 和 "b" 不能同时表示相同的字符串。编写一个方法判断 value 字符串是否匹配 pattern 字符串。

示例 1：

```
输入： pattern = "abba", value = "dogcatcatdog"
输出： true
```

示例 2：

```
输入： pattern = "abba", value = "dogcatcatfish"
输出： false
```

示例 3：

```
输入： pattern = "aaaa", value = "dogcatcatdog"
输出： false
```

示例 4：

```
输入： pattern = "abba", value = "dogdogdogdog"
输出： true
解释： "a"="dogdog",b="", 反之也符合规则
```

提示：

- `1 <= len(pattern) <= 1000`
- `0 <= len(value) <= 1000`
- 你可以假设 `pattern` 只包含字母 `"a"` 和 `"b"`， `value` 仅包含小写字母。

字符串暴力匹配。

基本想法是枚举a和b的长度la和lb，一定要满足lacnt(a)+lbcnt(b)=value.length()。

其中cnt()是某个字符的出现次数。

以上的枚举需要消耗O(n)的时间，剩下的就是匹配了。

注意需要找到b第一次出现的位置，在那儿截取一段lb长度的字符串下来作为b，代码如下：

```
                if(lb>0) {
                    int j=0;
                    for(auto i:pattern)
                        if(i=='a') j+=la;
                        else break;
                    if(j<value.length()) b=value.substr(j,lb);
                }
```

上面判断了是否存在b。这里我把a固定为第一个字符，代码如下：

```
        if(pattern[0]=='b')
            for(auto &i:pattern)
                i='a'+'b'-i;
```

注意判断a和b不能相等的条件。

除此之外还要考虑一些奇怪特判，代码如下：

```
        if(pattern==""&&value=="") return 1;
        if(pattern=="") return 0;
        if(value=="") return A*B==0;
```

主要有三种情况：

1. 两个都为空串，返回true。
2. 模式串为空而value不为空，返回false。
3. value为空，如果模式串中既存在a又存在b则返回false，否则把a或者b全部映射为空串就可以匹配上了。

面试题 10.01. 合并排序的数组

给定两个排序后的数组 A 和 B，其中 A 的末端有足够的缓冲空间容纳 B。 编写一个方法，将 B 合并入 A 并排序。

初始化 A 和 B 的元素数量分别为 *m* 和 *n*。

示例:

```
输入:
A = [1,2,3,0,0,0], m = 3
B = [2,5,6],       n = 3

输出: [1,2,2,3,5,6]
```

说明:

- `A.length == n + m`

归并排序的一部分

压行比较厉害，比如这种：

```
            tmp[k++]=min(A[i],B[j]);
            A[i]<=B[j]?++i:++j;
```

还有这种：

```
            while(i<m) tmp[k++]=A[i++];
```
