链滴

leetcode解题报告-字符串

作者：Zing

原文链接：https://ld246.com/article/1457012743311

来源网站：链滴

如果说链表让人又爱又恨的话，那么对字符串就只剩下恨了。个人认为，字符串题目是 leetcode 中最具备挑战性的，然而也是我们在日常编程中最常遇到的问题之一，因此学好字符串是非常重要的
字符串是题型最多变的考点，通常考察点有子串、回文、数学、编辑距离、括号问题、模式匹配类型转换、按要求格式化等等等等，采用的方法通常有栈、状态机、滑动窗口、哈希表、动态规划等通常解法灵活多变。我暂时还没能力像链表一样对字符串题目进行比较好的归类和总结，因此只是把 letcode 的题罗列在下。希望日后能有更深的理解，再来进一步总结。
<hr>
##3.Longest Substring Without Repeating Characters （子串）

难度：Medium

题意：
给定一个字符串，要求找出不重复的最长子串长度。

思路：
使用两个下标，每次尝试把 s[j]加入到字串 s[i:j]中，若 s[i:j]中不存在 s[j],则 j+=1,否则 i 向前推进到和 s[j]重复的索引的下一个。即，j 是每次 1 步向前推进，i 是每次跳跃到与 s[j]重复的下一个。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {string} s
      # @return {integer}
      def lengthOfLongestSubstring(self, s):
          if not s:return 0
          s = ''.join([s,'#'])
          i,j = 0,1
          max = 1
          while j < len(s):
              index = s[i:j].find(s[j])
              if index >= 0 or s[j] == '#':
                  if j - i > max:
                      max = j-i
                  i += index+1
              j += 1
          return max
</code></pre>

<hr>
##5.Longest Palindromic Substring （回文子串）

难度：Medium

题意：
给定字符串 s，找出最长回文字串。假定字符串最大长度为 1000，且仅存在唯一一个最长回文子串。
/p>

思路：
我这里挤破脑袋，只能想到最傻的办法，由于字符串最大长度只有 1000，所以时间上允许我这么做
遍历字符串，若该处可能成为回文串中心（奇和偶），则向两侧探索。
不要小看代码中的 <code> len(s)-it>len(subStr)/2</code>，这个条件能减掉 1 半的枝，使时
缩短 1 半。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
      # @param {str
ng} s
      # @return {str
ng}
      def longestPal
ndrome(self, s):
          if not s or l
n(s) == 1:
              return s
          subStr = ''
          it = 1
          s = ''.join(['~
,s])
          while it <
en(s)-1 and len(s)-it>len(subStr)/2:
              if not(s[it]
== s[it+1] or s[it-1] == s[it+1]):
                  it += 1
                  contin
e
              if s[it] ==
s[it+1]:
                  l,r = it,i
+1
                  subStr
= self.trylr(s,l,r,subStr)
              if s[it-1]
= s[it+1]:
                  l,r = it-
,it+1
                  subStr

= self.trylr(s,l,r,subStr)
            it += 1
        return subSt

    def trylr(self,s,l
r,sub):
        while l>0
and r<len(s) and s[l]==s[r]:
            l,r=l-1,r+

        if r-l-1 >
en(sbu):return s[l+1:r]
        else:return
ub
</code></pre>

思路 2：
思路一是最直观但也是最笨的办法。仔细思考，我们还得在思路 1 的基础上进行剪枝。回文串的处理大的难度之一在于奇串和偶串，我们可以进一步把回文串的格式概括为以下格式：sub1+[s]*n+sub2 意思是，回文串的中间是 n 个相同的字母（n 可以是奇数也可以是偶数，当然也可以是 1），左右两是镜像对称的两个字符串。那么我们在判断是否为回文串的时候，可以直接找到中间相同的字符，再两侧延伸。而且，更为关键的是，查找下一回文串的时候，可以直接跳过该中间重复字符（这个是最关键的剪枝）。
*代码 2：（参考大神的代码）
<pre><code class="highlight-chroma"><span class="highlight
cl"> def longestPalindrome(self, s):
    lenS = len(s)
    if lenS <= 1:
return s
    minStart, max
en, i = 0, 1, 0
    while i < len
:
        if lenS - i &lt
= maxLen / 2: break
        j, k = i, i
        while k <
enS - 1 and s[k] == s[k + 1]: k += 1
        i = k + 1
        while k <
enS - 1 and j and s[k + 1] == s[j - 1]: k, j = k + 1, j - 1
        if k - j + 1
gt; maxLen: minStart, maxLen = j, k - j + 1
    return s[minSt
rt: minStart + maxLen]
</code></pre>

<hr>
##6. ZigZag Conversion （格式化）

难度：Easy

题意：
给定一个字符串，以 ZigZag 形式排列（大概以“N”的形式），然后按行重新排列。

思路：
没有特殊的方法，只要把题目的意思看懂就 ok 了，按照题目要求去做就行，找到字符序号和位置的对关系。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {string} s
      # @param {integer} numRows
      # @return {string}
      def convert(self, s, numRows):
          if numRows == 1 or numRows >= len(s):
              return s
          zig = [[] for i in range(numRows)]
          if numRows == 2:
              return ''.join(s[::2]),''.join(s[1::2])]
          for i in rang(len(s)):
              rn = i%(2*numRows-2)
              if rn<numRows:
                  zig[i%(2*numRows-2)].append(s[i])
              else:
                  zig[2*numRows-2-rn].append(s[i])
          return ''.join(''.join(zig[i]) for i in range(numRows))
</code></pre>

<hr>
##8. String to Integer （类型转换）

难度：Easy

题意：
实现 atoi 把字符串转化为数字。

思路：
面试最爱出的题目之一，重点考察对特殊情况的覆盖和处理，空字符，‘+/-’，非数字，溢出等。<
p>

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
  # @param {strin
} str
  # @return {integ
r}
  def myAtoi(self, s
r):
      str = str.lstrip()
      if not str: retur
 0
      for i in range(l
n(str)):
          if not('0'&lt
=str[i]<='9' or str[i] in('+','-')):
              str = str[:i
]
              break
      try:
          f = 1
          intx = int(str

          if intx >=
2147483647:
              intx = 21
7483647
          elif intx <
 -2147483648:
              intx = -2
47483648
          return intx
      except:
          return 0
</code></pre>

<hr>
##10. Regular Expression Matching (模式匹配)

难度：Hard

题意：
实现支持'.'和''的正则表达式。'.'匹配任何一个字符，''匹配一个或多个之前的字符。给的表达式需要匹配整个字符串

思路：
翻回来，看到我自己的这个代码，也是无语了，我居然直接调用了 python 的 re 模块。。。。。。道题就是要我实现这个。。。

代码：
<pre><code class="highlight-chroma"> import re
  class Solution:
      # @param {string} s
      # @param {string} p
      # @return {boolean}
      def isMatch(self, s, p):
          pattern = re.compile(p)
          match = pattern.match(s)
          if match:
              if match.group() == s:
                  return true
          return False
</code></pre>

思路：
好吧这回好好地思考以下，毕竟是一道 Hard 题。这道题的最难点在于 <code>*</code>,因为其可匹配 0 个或多个字符，这使得直接遍历来匹配不可能实现。换个角度，递归行不行？这道题剪枝行不？都还不行，那就是动态规划了。我认为，DP 是水平的分界线：完全不会 DP 的人，看得懂 DP 的，可以设计 DP 的人。
言归正传，我们以 <code>dp[i][j]=True</code>,表示前 i 个字母（s[0:i]）可以匹配前 j 个模式(p[0:j])。

若 p[j-1]不是 <code>*</code>,意味着在 dp[i-1][j-1]=True 的前提下，p[j-1]必须是 <code>./code> 或等于 s[i-1]，才可能匹配。
若 p[j-1]是 <code>*</code>:

dp[i][j-2]匹配，<code>p[j-2]*</code> 表示 0 个字符
dp[i-1][j]匹配，<code>p[j-2]*</code> 多表示 1 个 s[i-1],则 p[j-2]必须是 s[i-1]或 <code>.</ode>

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {str ng} s
      # @param {str ng} p
      # @return {bo lean}
      def isMatch(se f, s, p):
          lens,lenp = en(s),len(p)
          dp = [[False *(lenp+1) for i in range(lens+1)]
          dp[0][0]=Tr e
          for j in rang (2,lenp+1):
              dp[0][j] = dp[0][j-2] and p[j-1]=='*'

          for i in rang (1,lens+1):
              for j in ra ge(1,lenp+1):
                  if p[j-1 =='*':
                      dp[i] j] = dp[i][j-2] or dp[i-1][j] and p[j-2] in (s[i-1],'.')
                  else:
                      dp[i] j]=dp[i-1][j-1] and p[j-1] in (s[i-1],'.')
          return dp[le s][lenp]
</code></pre>

<hr>
##44. Wildcard Matching （模式匹配）

难度：Hard

题意：
实现支持 <code>?</code> 和 <code>*</code> 的通配符匹配模式。
<code>?</code> 匹配任意单个字符
<code>*</code> 匹配任意字符串序列，包括 0 个字符。注意和第 10 题的区别。

思路：
T.T。。。我居然变着法去调用 re 模块，然后也是通过了。。。。把 <code>?</code> 替换成 <co

e>.</code>，把字符串以*分隔，然后分段用 re 去匹配。这份代码又臭又长，请直接看思路 2。</p

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> import re
  class Solution:
      # @param {str
ng} s
      # @param {str
ng} p
      # @return {bo
lean}
      def isMatch(se
f, s, p):
          if p == s:
              return Tr
e
          if not p:
              return Fal
e
          p = p.replac
('?','.')
          ##split p by
*'
          plist = p.spli
('*')
          begin = 0
          for i,psplit in
enumerate(plist):
              if '.' in psp
it:
                  ##last
ne
                  if i ==
en(plist)-1:
                      begi
 = self.regularMatch(s,psplit,begin,1)
                      if p[
1]!='*' and begin+len(psplit) !=len(s):
                          re
urn False
                  else:
                      begi
 = self.regularMatch(s,psplit,begin,0)
                  if begin
== -1:
                      retu
n False
                  ## first
one
                  if i==0
and p[0] != '*' and begin!= 0:

```
                return False
            begin = len(psplit)
        else:
            ##last one
            if i == len(plist)-1:
                lb = s.rfind(psplit)
                if lb &lt; begin:
                    return False
                else:
                    begin = lb
                    if [-1]!='*' and begin+len(psplit) !=len(s):
                        return False
            else:
                begin = s.find(psplit,begin)
                if begin == -1:
                    return False
                ##first one
                if i==0 and p[0] != '*' and begin!= 0:
                    return False
                begin = len(psplit)
        return True

    def regularMatch(self, s, p,begin,islast):
        pattern = re.compile(p)
        result = pattern.search(s,begin)
        if not result:
            return 1
        if not islast:
            return result.start()
        if islast:
```

   while r
sult:
   begi
 = result.start()
   resul
 = pattern.search(s,begin+1)
   return
egin
</code></pre>

<p>思路 2：我是不是有点不按套路出牌啊。正经来说，我们应该考虑以下动态规划的解法。我们以
code>dp[i][j]=True</code>,表示前 i 个字母（s[0:i]）可以匹配前 j 个模式(p[0:j])。</p>

若 p[j-1]不是 <code>*</code>,意味着在 dp[i-1][j-1]=True 的前提下，p[j-1]必须是 <code>
</code> 或等于 s[i-1]，才可能匹配。
若 p[j-1]是 <code>*</code>:
存在 dp[k][j-1]=True，1<=k<=i。因为*可以与任意 s[k：i]匹配

<p>奇怪，搞不懂，代码报超时的 Testcase，我跑了以下均在 100ms 以内。
DP 通常会有多种解决方法，关键在于 DP 如何定义。这道题还可以使用一维 dp 能解决。</p>

<p>代码：</p>
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
   # @param {str
ng} s
   # @param {str
ng} p
   # @return {bo
lean}
   def isMatch(se
f, s, p):
   lens,lenp =
en(s),len(p)
   if lenp-p.co
nt('*')>lens:return False
   dp = [[False
*(lenp+1) for i in range(lens+1)]
   dp[0][0]=Tr
e
   for j in rang
(1,lenp+1):
   dp[0][j] =
dp[0][j-1] and p[j-1]=='*'

   for i in rang
(1,lens+1):
   for j in ra
ge(1,lenp+1):
<span class="highlight-line"><span class="highlight-cl">   if p[j-1]

='*':
                    dp[i]
j]=dp[i-1][j-1] and p[j-1] in (s[i-1],'?')
                else:
                    flag
= 0
                    for k
n range(i,-1,-1):
                        if
p[k][j-1]:

dp[i][j],flag=True,1

break
                    if no
 flag:
                        d
[i][j]=False

        return dp[le
s][lenp]
</code></pre>

<hr>
##12. Integer to Roman （类型转换）

难度：Medium

题意：
给一个数字，转化成罗马数字表示。输入数字范围在 1 到 3999 之间。

思路：
先了解以下罗马数字的表示法：I 代表 1，V 代表 5，X 代表 10，L 代表 50，C 代表百，D 代表 500
M 代表 1000。放在更高阶符号左边表示减，右边表示加，不允许出现 3 个以上同阶符号。这个代码
可避免会比较长，但是思路确是很简单。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param {int
ger} num
    # @return {str
ng}
    def intToRom
n(self, num):
        roman=''
        roman = ''.j
in([roman,'M'*(num//1000)])
        num %= 10

```
        if num//10
&lt;4:
            roman = ''
.join([roman,'C'*(num//100)])
        elif num//1
0==4:
            roman = ''
.join([roman,'CD'])
        elif num//1
0&lt;9:
            roman = ''
.join([roman,'D','C'*(num//100-5)])
        else:
            roman = ''
.join([roman,'CM'])
        num %= 10

        if num//10&
t;4:
            roman = ''
.join([roman,'X'*(num//10)])
        elif num//1
==4:
            roman = ''
.join([roman,'XL'])
        elif num//1
&lt;9:
            roman = ''
.join([roman,'L','X'*(num//10-5)])
        else:
            roman = ''
.join([roman,'XC'])
        num %= 10
        if num&lt;4:
            roman = ''
.join([roman,'I'*num])
        elif num==4

            roman = ''
.join([roman,'IV'])
        elif num&lt;
:
            roman = ''
.join([roman,'V','I'*(num-5)])
        else:
            roman = ''
.join([roman,'IX'])
        return rom
n
```


<hr>

##13. Roman to Integer （类型转换）

难度：Easy

题意：
给一个字符串表示的罗马数字，转化阿拉伯数字表示。输入数字范围在 1 到 3999 之间。

思路：
和上一题刚好是相反，然而确是简单了许多。若当前符号不大于前一个符号，直接加上，相反则需要去之前那个符号。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {str ng} s
      # @return {int ger}
      def romanToIn (self, s):
          ints = pre = 0
          mp = {'I':1,'V :5,'X':10,'L':50,'C':100,'D':500,'M':1000}
          for i in rang (len(s)):
              if mp[s[i] <=pre:ints += mp[s[i]]
              else:ints = ints - 2*pre + mp[s[i]]
              pre = mp s[i]]
          return ints
</code></pre>

<hr>
##14. Longest Common Prefix （子串）

难度：Easy

题意：
给定一个字符串数组，找出最长公共前缀

思路：
由于是公共前缀，只需要以第一个字符串的前缀去匹配其他字符串即可。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {string[]} strs
      # @return {string}
      def longestCommonPrefix(self, strs):
          if not strs:return ''
          if len(strs)==1:return strs[0]
          for i in rang(1,len(strs[0])+1):
              for j in range(1,len(strs)):
                  if strs[j]find(strs[0][:i])!=0:
                      if no i:return ''
                      else:eturn strs[0][:i-1]
          return strs[0]

</code></pre>

<hr>
##17. Letter Combinations of a Phone Number （格式化）

难度：Medium

题意：
手机九宫格，输入一个字符串代表的数字，输出所有可能的字母组合。

思路：
最直观的思路就是做笛卡尔乘积即可。
这里也可用递归回溯来实现。题目比较简单就不单独实现了。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {string} digits
      # @return {string[]}
      def letterCombinations(self, digits):

```
        digitsMap={
'2':['a','b','c'],
                '3':['d'
'e','f'],
                '4':['g'
'h','i'],
                '5':['j',
k','l'],
                '6':['
','n','o'],
                '7':['p'
'q','r','s'],
                '8':['t',
u','v'],
                '9':['w
,'x','y','z'],
                '1':[],
0':[]}
        result = []
        for i in digits

            result = s
lf.mass(result,digitsMap[i])
        return result

    def mass(self,li
ta,listb):
        if not lista:re
urn listb
        if not listb:r
turn lista
        result = []
        for a in lista:
            for b in lis
b:
                result.
ppend(''.join([a,b]))
        return result
```


<hr>
##20. Valid Parentheses（括号问题）

难度:Easy

题意：
给定一个字符串，只包含 <code>(){}[]</code>，检测该字符串是否有效，即括号是否能匹配。

思路：
使用栈，当是左括号时，压入栈，当是右括号时，弹出栈顶，检查是否匹配。

代码：
<pre><code class="highlight-chroma"> class Solution:
    # @param {string} s
    # @return {boolean}
    def isValid(self, s):
        m = {'(':')','[':']','{':'}'}
        stacklist = []
        for p in s:
            if p in('(','{','['):
                stacklis.append(p)
            else:
                if not stacklist or p != m[stacklist[-1]]:
                    return False
                else:
                    stacklist.pop()

        if not stacklist:return True
        else:return False
</code></pre>

<hr>
##22. Generate Parentheses （括号问题）

难度：Medium

题意：
给定 n 对小括号，要求写出所有合法的形式。

思路：
由于都是小括号，在左边是完整模式时，下一个必须是左括号。在以上前提下，只要把 n 个左括号和 n 个右括号全部消费完即可。使用递归，记录目前左括号和右括号的消费情况。

代码：
class Solution:
# @param {integer} n

# @return {string[]}
def generateParenthesis(self, n):
self.result = []
left = right = n
self.generate('',left,right)
return self.result
<pre><code class="highlight-chroma"><span class="highlight
cl"> def generate(self,seq,left,right):
      if left == right
== 0:
          self.result.a
pend(seq)
      elif left == rig
t:
          self.generat
('').join([seq,(']),left-1,right)
      elif left < ri
ht:
          if left:
              self.gene
ate('').join([seq,'(']),left-1,right)
          if right:
              self.gene
ate('').join([seq,')']),left,right-1)
</code></pre>

<hr>
##28. Implement strStr() （子串）

难度：Easy

题意：
实现 strStr()，也就是寻找子串在母串中的第一次出现位置，若无出现返回-1

思路：
也就是让我们实现 python 里的 find 函数，然而我又一次直接调用 find 然后 AC 了。这道题是典型查找字串问题，使用暴力方法时间复杂度是 O(m*n)。当然我们也不会忘记算法课那个让我们头疼的 MP 算法，时间复杂度是 O(m+n)。这里我先实现暴力方法，用两个下标太麻烦，我干脆直接用了递。实现完之后发现这种暴力的方法和 python 的 find 时间是基本一样的，都是 40+ms，这么优秀的杂度，说明这道题没有必要去实现 KMP（这可是一道 Easy）。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
      # @param {str
ng} haystack
      # @param {str
ng} needle
      # @return {int

ger}
    def strStr(self, haystack, needle):
        ##return haystack.find(needle)
        lenN,lenH=ln(needle),len(haystack)
        if lenN>lnH:
            return -1
        elif not needle or haystack[:lenN]==needle:
            return 0
        else:
            res = self.trStr(haystack[1:],needle)
            return 1+res if res!=-1 else -1
</code></pre>

<hr>
##30. Substring with Concatenation of All Words （子串）

难度：Hard

题意：
给定一个字符串 s，及一个单词表，单词表中的每个单词长度均相同，要求找出 s 的每个子串的 inde，子串满足：由单词表中所有单词连接构成，且单词之间不相互交叉。

思路：
使用两个下标 i、j,i 表示子串开始位置，然后判断 s[j:j+len(word)]是否存在单词表中，由于限制单词间不能相互交叉，因此在匹配的情况下，j 每次推进 len(word)。又要求包含每个单词，且相同单词能出现多次，因此我们使用 map 来记录，当前子串还需覆盖的单词及数量。

代码：
<pre><code class="highlight-chroma"> class Solution:
    # @param {string} s
    # @param {string[]} words
    # @return {integer[]}
    def findSubstring(self, s, words):
        result,wordir = [],{}
        if not s or not words:

```
            return result
        for w in words:
            if wordDict.get(w):
                wordDict[w] += 1
            else:
                wordDict[w] =1
        i = j =0
        slen,wordlen = len(s),len(words[0]),len(words[0])*len(words)
        while i+wordslen&lt;= slen:
            tag = wordDict.copy()
            j = i
            while j+wordlen &lt;= slen:
                count = tag.get(s[j:j+wordlen])
                if count and count != 1:
                    tag[sj:j+wordlen]] -= 1
                    j += wordlen
                elif cnt and count ==1:
                    del tg[s[j:j+wordlen]]
                    j += wordlen
                else:
                    brea

            if not tag:
                result.ppend(i)
            i+=1
        return result
```


<hr>
##32. Longest Valid Parentheses (括号，子串)

难度：Hard

题意：

给定一个只包含 <code>(</code> 和 <code>)</code> 的字符串，要求找出最长的括号合法的子长度。

思路：
这道题我尝试了很多种思路，若按照常规思路，最难判断的是我如何向前推进去探索这个字符串，若每次 1 步肯定超时。因此还是老办法：常规思路？-> 递归？-> 剪枝？->DP。没错了这道就是 DP。
我们记 dp[i]=n 表示以 s[i]为结尾的字符串的最长合法子串为 n。若当前字符是 <code>)</code> 当前字符左侧是一个合法子串，合法子串的左边是 <code>)</code>,则有如下关系：<code>dp[i]=p[i-1]>0</code>,最后还需把两个连续和合法串连接起来。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {string} s
      # @return {integer}
      def longestValidParentheses(self, s):
          if not s:return 0
          dp = [0]*len(s)
          for i in rang(1,len(s)):
              if s[i] == ')':
                  if i-1-dp[i-1]>= 0 and s[i-1-dp[i-1]] == '(':
                      dp[i] = dp[i-1]+2
                      if i-1-dp[i-1]>0:
                          dp[i] = dp[i] + dp[i-1-dp[i-1]-1]
          return max(dp)
</code></pre>

<hr>
##38. Count and Say （格式化）

难度：Easy

题意：
给定一个“数-说”数字序列，其中第一个数是 1，每个数字是前一个数字的表达：
c 个 n

思路：
常规思路，关键是要看懂问题，不是输入一个数字输出该数字的表达。而是问你以 1 开始的序列的第 个数字是多少。若是采用 int 型直接处理会比较麻烦，转为 string 类型会简单很多。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {int ger} n
      # @return {str ng}
      def countAnd ay(self, n):
          if not n:retu n
          strn,result = str(n),'1'
          for i in rang (n,1,-1):
              result = s lf.countPre(result)
          return result

      def countPre(s lf, pre):
          result,coun ='',1
          for i in rang (1,len(pre)+1):
              if i == len pre) or pre[i] != pre[i-1]:
                  result ''.join([result,str(count),pre[i-1]])
                  count 1
              else:
                  count = 1
          return result
</code></pre>

<hr>
##43. Multiply Strings （数学）

难度：Medium

题意：
给定两个字符串格式的数字，要求以字符串的格式返回两个数字的乘积。

思路：
直接把两个数字转成 int，乘积再转会 string 不就行了吗？确实是可以。但是你太年轻了，没有猜透题人的想法。这道题是考察字符串的 Medium 难度的题。这道题你需要像小学乘法一样一位一位去，然后相加，计算进位。。。。然而我认为这道题意义不大，所以我直接转 int 了。T.T，原谅我

代码：
<pre><code class="highlight-chroma"> class Solution:
  # @param {strin
} num1
  # @param {strin
} num2
  # @return {string

  def multiply(self,
num1, num2):
      return str(int(
um1)*int(num2))
</code></pre>

<hr>
##49. Group Anagrams （哈希）

难度：Medium

题意：
给定一个字符数组，将字母完全相同的分为一组。

思路：
将每个字符串进行排序，然后放在 hash 表中，key 是排序后的字符串，value 是排序前字符串列表。
/p>

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {str
ng[]} strs
      # @return {str
ng[]}
      def anagrams(
elf, strs):
          strsDir = {}
          result = []
          for i,str in e
umerate(strs):
              tmp = "".
oin((lambda x:(x.sort(),x)[1])(list(str)))

            if not str
Dir.get(tmp):
                strsDir[
mp] = []
            strsDir[t
p].append(i)
        for value in
trsDir.values():
            if len(val
e) >1:
                for ind
x in value:
                    resul
.append(strs[index])

        return result
</code></pre>

<hr>
##58. Length of Last Word （字符串分割）

难度：Easy

题意：
给定一个字符串，字符串包含大小写字母和空格，求最后一个单词的长度。

思路：
简单题，将字符串以空格分割，返回最后一个单词长度即可。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param {str
ng} s
    # @return {int
ger}
    def lengthOfL
stWord(self, s):
        strlist = s.spl
t()
        return 0 if n
t strlist else len(strlist[-1])
</code></pre>

<hr>
##65. Valid Number （状态机）

难度：Easy

题意：
判断给定字符串是否为数字

思路：
这答题最大的难度在于将所以的情况覆盖完全，空格，正负号，数字，非数字，点，科学计数法。使状态机解决该题目。只要把各种状态转移弄清楚了，写出状态转移表，这道题就迎刃而解了。状态机办法别人告诉你怎么画，必须自己耐心细心地画一遍才是自己的。

代码：
<pre><code class="highlight-chroma">class Solution:
    # @param {string} s
    # @return {boolean}
    def isNumber(self, s):
        inputType = [[' '],['+','-'],
                     ['0','1','2','3','4','5','6','7','8','9'],
                     ['.'],['e','E']]
        transitionTable=[[0,1,2,3,-1,-1],
                         [-1,-1,2,3,-1,-1],
                         [-1,-1,2,4,6,-1],
                         [-1,-1,5,-1,-1,-1],
                         [-1,-1,5,-1,6,-1],
                         [-1,-1,5,-1,6,-1],
                         [-1,7,8,-1,-1,-1],
                         [-1,-1,8,-1,-1,-1],
                         [-1,-1,8,-1,-1,-1],
                         [-1,-1,-1,-1,-1,-1]]
        state = 0
        for c in s:
            if state == -1:
                return False</code></pre>

            f = 0
            for i in ra
ge(5):
                if c in i
putType[i]:
                    state
= transitionTable[state][i]
                    f = 1
                    brea

            if f == 0:
                state =
-1
        if state in(2
4,5,8,9):
            return Tr
e
        else:
            return Fal
e
</code></pre>

<hr>
##67. Add Binary （数学）

难度：Easy

题意：
给定两个字符串表示的二进制数，同样以二进制字符串的形式返回俩数之和。

思路：
1.从后往前按位加，记录进位。和求两个链表表示的数字和思路一致。
2.转为 10 进制数字求和再转为 2 进制字符串。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param {str
ng} a
    # @param {str
ng} b
    # @return {str
ng}
    def addBinary(
elf, a, b):
        return bin(i
t(a,2)+int(b,2)).split('b')[1]
</code></pre>

<hr>
##68. Text Justification （格式化）

难度：Hard

题意：
给定一个单词表和长度 L，将单词进行格式化，使每一行长度为 L，且按要求格式化。每一行应该放尽可能多的单词，需要的时候填充空格使每一行恰好长度为 L。添加的空格应该尽可能平均分布在任两个单词之间，若无法平均分布，则左边的空格必须多余右边的空格。最后一行，必须左对齐，且单之间不允许填充额外的空格。

思路：
把题目中所说的规则看明白就费了半天。使用一个单词列表代表一行，用一个变量记录列表中所有单的长度之和，若单词加入之后不超出长度则加入列表，否则不加入，计算可以补空格的空缺数（列表度-1，注意只有一个单词时也是 1），计算每个空缺可以补充空格个数，然后插入到列表中合适的位，最后把列表中的单词按顺序组合起来即可。注意最后一行需要特殊处理。我这种思路没有什么难度就是处理逻辑比较繁琐。

代码：
<pre><code class="highlight-chroma"> class Solution:
      # @param {string[]} words
      # @param {integer} maxWidth
      # @return {string[]}
      def fullJustify(self, words, maxWidth):
          groups,grop =[],[]
          width = 0
          for word in words:
              width += len(word)
              if width + len(group) > maxWidth:
                  pos = ln(group) if len(group)==1 else len(group)-1
                  space (maxWidth-width+len(word))//pos
                  spaceLft = (maxWidth-width+len(word))% pos
                  for i in ange(1,2*pos,2):
                      if spceLeft:
                          g
</code></pre>

oup.insert(i,' '*(space+1))
                    s
aceLeft -= 1
                else:
                    g
oup.insert(i,' '*(space))
                groups
append(group)
                group
 []
                width
 len(word)
            group.ap
end(word)
        pos = len(g
oup) if len(group)==1 else len(group)-1
        space = (m
xWidth-width)
        for i in rang
(1,2*pos,2):
            if space:
                group.
nsert(i,' ')
                space-
1
        group.insert
len(group),' '*space)
        groups.app
nd(group)
        return [''.joi
(groups[i]) for i in range(len(groups))]
</code></pre>

<hr>
##71. Simplify Path （栈）

难度：Medium

题意：
以 Unix 风格给出一个绝对路径，要求简化路径。<code>/.</code> 表示当前路径；<code>/..</code> 表示上层路径

思路：
将字符串以 <code>/</code> 分割（注意如果你是 java 你需要转义 s.split("////")）。使用栈记录路径，若是 <code>.</code> 直接跳过，<code>..</code> 弹出栈顶，其他直接入栈，最后将栈内录连接起来即可。

代码：
<pre><code class="highlight-chroma"><span class="highlight

```
cl">  class Solution:
</span></span><span class="highlight-line"><span class="highlight-cl">      # @param {str
ng} path
</span></span><span class="highlight-line"><span class="highlight-cl">      # @return {str
ng}
</span></span><span class="highlight-line"><span class="highlight-cl">      def simplifyPa
h(self, path):
</span></span><span class="highlight-line"><span class="highlight-cl">          stack = []
</span></span><span class="highlight-line"><span class="highlight-cl">          for p in path
split('/'):
</span></span><span class="highlight-line"><span class="highlight-cl">              if not p or
p == '.':
</span></span><span class="highlight-line"><span class="highlight-cl">                  contin
e
</span></span><span class="highlight-line"><span class="highlight-cl">              elif p == '
.':
</span></span><span class="highlight-line"><span class="highlight-cl">                  if stack:
</span></span><span class="highlight-line"><span class="highlight-cl">                      stack
pop(-1)
</span></span><span class="highlight-line"><span class="highlight-cl">              else:
</span></span><span class="highlight-line"><span class="highlight-cl">                  stack.a
pend(p)
</span></span><span class="highlight-line"><span class="highlight-cl">          return '/' + '
'.join(stack)
</span></span></code></pre>
</li>
</ul>
<hr>
<p>##72. Edit Distance （编辑距离）</p>
<ul>
<li>
<p>难度：Hard</p>
</li>
<li>
<p>题意：<br>
给定两个单词，求从单词 1 转化为单词 2 的最少步骤数。允许的操作是：插入一个字符，删除一个字
，替换一个字符。</p>
</li>
<li>
<p>思路：<br>
最短编辑距离，超级经典的 DP 问题。我们定义 dp[i][j]表示 word1[:i]到 word2[:j]的编辑距离。则
如下关系：</p>
<ul>
<li>若 word1[i-1]和 word[j-1]相同，则有 dp[i][j]=dp[i-1][j-1]</li>
<li>若 word1[i-1]和 word[j-1]不相同，则有 3 中解决办法，在 word1[:i-1]到 word2[：j-1]编辑距
基础上，把 word1[i-1]改为 word[j-1]；在 word1[:i]到 word2[:j-1]编辑距离基础上，在 word1 末
删除一个字符；在 word1[:i-1]到 word2[:j]编辑距离的基础上，在 word1 末尾增加一个字符。</li>
</ul>
</li>
<li>
<p>代码：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">  class Solution:
</span></span><span class="highlight-line"><span class="highlight-cl">      # @param {str
```

```
ng} word1
        # @param {str
ng} word2
        # @return {int
ger}
    def minDistan
e(self, word1, word2):
        d = [[0 for j
n range(len(word2)+1)]for i in range(len(word1)+1)]
        for i in rang
(len(word1)+1):
            for j in ra
ge(len(word2)+1):
                if i ==
:
                    d[i][j
 = j
                elif j =
 0:
                    d[i][j
 = i
                elif wo
d1[i-1] == word2[j-1]:
                    d[i][j
 = d[i-1][j-1]
                else:
                    d[i][j
 = min(d[i-1][j-1],d[i][j-1],d[i-1][j])+1
        return d[-1][
1]
```

##115. Distinct Subsequences（编辑距离）

- 难度：Hard
- 题意：
给定字符串 S 和 T，要求只通过删除字符的形式将 S 转化为 T 的方法数。
- 思路：
这道题跟最短编辑距离基本上是一样的，不过编辑方法限制为只能删除。因此还是用动态规划。我们义 dp[i][j]表示：s[:i]转化为 t[:j]的方法数。则有以下关系：
  - 删除 s[i-1]，dp[i][j]=dp[i-1][j]，这一项必有
  - 如果 s[i-1]=t[j-1]，则保留 s[i-1]，有 dp[i][j]=dp[i-1][j]+dp[i-1][j-1]
- 代码：

<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution(object):
   def numDistin
t(self, s, t):
   """
   :type s: str
   :type t: str
   :rtype: int
   """
   if not(s and
):return 0
   c1,c2 = len(s
,len(t)
   if c1<c2:r
turn 0
   dp=[[0 for i
n range(c2+1)] for j in range(c1+1)]
   for i in rang
(c1+1):dp[i][0] = 1
   for i in rang
(1,c1+1):
   for j in ra
ge(1,c2+1):
   dp[i][j]
dp[i-1][j]
   if s[i-1
==t[j-1]:
   dp[i]
j]+=dp[i-1][j-1]
   return dp[-1
[-1]
</code></pre>

<hr>
<p>##76. Minimum Window Substring（子串）</p>

<p>难度：Hard</p>

<p>题意：
给定字符串 T 和 S，找到字符串 S 包含 T 中全部字符的最短子串。</p>

<p>思路：
T 串放到哈希表中，用来判断是否完全覆盖。使用滑动窗口来寻找子串，左游标直接跳过没在哈希表的字符，右游标向前推进直到哈希表中所有值均为 0。注意左游标移动的时候需要恢复现场。代码写比较凌乱。</p>

<p>代码：</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl"> class Solution:

```
        # @param {string} s
        # @param {string} t
        # @return {string}
        def minWindow(self, s, t):
            result = ''
            if not t:return result
            dicT = {}
            for char in t:
                if not dicTget(char):
                    dicT[char] = 0
                dicT[char] += 1
            i=j=0
            while i&lt;len(s) and  s[i] not in t:
                i += 1
            s = s[i:]
            i = 0
            while j &lt;ln(s) or max(dicT.values())&lt;=0:
                if max(diT.values())&lt;=0:
                    ##getindow
                    if not result:
                        result = s[i:j]
                    elif j-i&lt; len(result):
                        result = s[i:j]
                    dicT[s[i] += 1
                    i += 1
                    while i&lt;len(s) and  s[i] not in t:
                        i += 1
                    contine
                while j&lt;len(s) and s[j] not in t:
                    j += 1
                if j&lt;lens):
```

                dicT[s[j
] -= 1
                j += 1
        return result
</code></pre>

<hr>
##87. Scramble String （树）

难度：Hard

题意：
通过将字符串分割成两个非空的部分构成一个二叉树。选择任意非叶子节点，交换其左右子节点位置
求问字符串 s1 能否通过这种形式转化为 s2

思路：
看到凡是跟树有关的题，应该往递归上靠一下，因为树的定义本身就是递归的。只要能想到用递归解
这道题就完成一大半了。s1 是 s2 的变换，则有从某处分割，s1 的左子串是 s2 左子串的变换且 s1
子串是 s2 右子串的变换，或 s1 左子串是 s2 右子串的变化且 s1 右子串是 s2 左子串的变化。这个检
过程本身也是递归的。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution(object):
    def isScramble
self, s1, s2):
        """
        :type s1: str
        :type s2: str
        :rtype: bool
        """
        length = len
s1)
        if length!=l
n(s2):return False
        if s1==s2:re
urn True
        count = [0]
26zhu'yi
        for c in s1:
            count[ord
c)-97]+=1
        for c in s2:
            count[ord
c)-97]-=1
        for c in cou
t:
            if c!=0:re
urn False

</code></pre>

        for i in rang
(1,length):
            if (self.isS
ramble(s1[:i],s2[:i]) and self.isScramble(s1[i:],s2[i:])) or (self.isScramble(s1[:i],s2[length-i:]) and se
f.isScramble(s1[i:],s2[:length-i])):
                return
rue
        return False
</code></pre>

<hr>
##91. Decode Ways （格式化）

难度：Medium

题意：
有一个包含字母 A-Z 的信息，使用以下编码方式 A->1,B->2...Z->26。给定一个编码后的
息，求出其所有可能的解码种类，如 12 可解码为 AB 或 L。

思路：
这是一道比 Hard 通过率还低的 Medium 题，因为这是一道 DP 题，说明大家对 DP 的理解还是不
深，找个机会把我之前学习 DP 的笔记整理再发出来（又是一个坑）。我们定义 dp[i]=n 表示表示 s[-i
]的可能的解码数，为什么要从后往前遍历，一会你就可以体会到。则有如下关系：

若 s[-i]=='0',则解码数为 0（0 无对应解码）
若 s[-i]!='0',则可以 s[-i]单独解码的情况 + 若 s[-i:-i+2]可解码两种可能情况。
相同的思路，如果以从前往后遍历，则需要分析的情况会复杂许多。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param {str
ng} s
    # @return {int
ger}
    def numDecod
ngs(self, s):
        if not s: retu
n 0
        count = [1]
        if s[-1] == '0
:
            count.ap
end(0)
        else:
            count.ap
end(1)
        for i in rang

(2,len(s)+1):
            c = 0 if s[-
]=='0' else count[-1]
            if '10'<
s[-i:][0:2]<='26':
                print(s[
i:][0:2])
                c += c
unt[-2]
            count.ap
end(c)
        return count
-1]
</code></pre>

<hr>
##93. Restore IP Addresses（格式化）

难度：Medium

题意：
给定一个字符串表示的数字，要求将其转化为所有可能的 IP 地址。

思路：
IP 地址由 4 个小于 256 的整数构成，注意不允许连续 0。这道题使用递归可以很容易解决。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param {str
ng} s
    # @return {str
ng[]}
    def restoreIpA
dresses(self, s):
        self.result =
]
        self.reduce(s
[])
        return self.r
sult
    def reduce(self
s0,ele0):
        s = s0[::]
        ele = ele0[::]
        if len(ele)=
4 and not s:
            self.result
append('.'.join(ele))

            elif len(ele)
lt;4 and s:
                for i in ra
ge(1,len(s)+1):
                    if int(s[:
])<256:
                        if len
s[:i])>1 and s[0]=='0':
                            br
ak
                        ele.
ppend(s[:i])
                        self.
educe(s[i:],ele)
                        ele.
op(-1)
                    else:
                        brea

</code></pre>

<hr>
##97. Interleaving String （子串）

难度：Hard

题意：
给定 s1，s2，s3，求 s3 是否由 s1 和 s2 交织得到。

思路：
题意是要求在 s3 中 s1 和 s2 中的字符的相对顺序保持不变。这道题的难点在于当一个字符同时出现在 s1 和 s2 中时，如何判断是属于谁。若用递归回溯，直接就超时了，题目的规模过大。因此启用固定题法，递归回溯走不同的时候就 DP。
我们定义 dp[i][j]=True 表示 s1[:i]和 s2[:j]可以交织成 s[:i+j]，则有如下关系：

若 s1[i]==s3[i+j+1],则有 dp[i+1][j+1]=dp[i][j+1],(最后一个字符由 i 来出)
若 s2[j]==s3[i+j+1],则有 dp[i+1][j+1]=dp[i+1][j],(最后一个字符由 j 来出)
注意这两者是或的关系，有可能均相等或均不相等的情况。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution(object):
      def isInterleav
(self, s1, s2, s3):
          """
          :type s1: str
          :type s2: str
          :type s3: str

        :rtype: bool
        """
        if not s2:ret
rn s1==s3
        elif not s1:re
urn s2==s3
        len1,len2=l
n(s1),len(s2)
        if len1+len2
=len(s3):return False
        dp = [[False
or i in range(len2+1)] for j in range(len1+1)]
        dp[0][0] = T
ue
        for i in rang
(len2):
            dp[0][i+1]
= (dp[0][i] and s2[i]==s3[i])
        for i in rang
(len1):
            dp[i+1][0]
= (dp[i][0] and s1[i]==s3[i])
        for i in rang
(len1):
            for j in ra
ge(len2):
                dp[i+1]
j+1] = ((dp[i][j+1] and s1[i]==s3[i+j+1]) or

  (dp[i+1][j] and s2[j]==s3[i+j+1]))
        return dp[le
1][len2]
</code></pre>

<hr>
##125. Valid Palindrome （回文）

难度：Easy

题意：
给定一个字符串，判断其是否为回文串，不考虑大小写及空格和标点

思路：
去掉空格和标点，转化为小写字母，然后将字符串逆序比较。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
      # @param {str

ng} s
     # @return {bo
lean}
     def isPalindro
e(self, s):
         slist = []
         for c in s:
             if c.isalph
() or c.isdigit():
                 slist.ap
end(c.lower())
         return slist
=slist[::-1]
</code></pre>

<hr>
##127. Word Ladder （编辑距离）

难度：Medium

题意：
给定两个单词 beginWord 和 endWord，已经一个单词表，求通过单词表中的词从 beginword 到 e
dWord 的最短变化序列的长度。变化要求每次只能改变一个字符。

思路：
这道题其实也是编辑距离的变种，编辑时限制只能改变字符，且改变必须在给定的单词表范围内。然
我们这次不使用 dp，原因是在没有限制时，每一步编辑的结果是无限的，现在加上限制只有，每一
编辑的结果是可列举的，我们可以把所有单词的变化现象成一棵树。要求最短距离，基本固定模式就
按层次遍历，第一次到达 endWord 时的层次就是最短编辑距离。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution(object):
     def ladderLen
th(self, beginWord, endWord, wordDict):
         """
         :type begi
Word: str
         :type endW
rd: str
         :type wordD
ct: Set[str]
         :rtype: int
         """
         letter=[chr(
7+i) for i in range(26)]
         queue=[beg
nWord]
         dic={begin

```
ord:1}
        while queue

            word = queue.pop(0)
            level = dic[word]
            for i in range(len(word)):
                tmpword = list(word)
                for j in letter:
                    if tpword[i]==j:continue
                    tmpord[i]=j
                    tmp = ''.join(tmpword)
                    if tp==endWord:return level+1
                    if tp in wordDict and tmp not in dic:
                        queue.append(tmp)
                        dic[tmp]=level+1

        return 0
```


<hr>
##126. Word Ladder II （编辑距离）

难度：Hard

题意：
与 127 题意基本一样，不同的是要求返回所有最短序列的具体路径

思路：
我采用的思路还是和 127 题一样，使用队列按层次遍历，当时同时需要注意的是，遍历的同时需要记父节点。且由于要求返回所有最短，因此最短那一层必须完全遍历，而不是像 127 一样遇到 endword 即可。

代码：

```
 class Solution(object):
    def findLadders(self, beginWord, endWord, wordlist):
```

```
        """
        :type beginWord: str
        :type endWord: str
        :type wordlist: Set[str]
        :rtype: List[List[int]]
        """
        letter=[chr(97+i) for i in range(26)]
        queue=[beginWord]
        dic={beginWord:1}。
        self.father={beginWord:{}}
        self.res=[]
        end=2**31
        while queue:
            word = queue.pop(0)
            level = dic[word]
            if level>end:
                break
            for i in range(len(word)):
                tmpword = list(word)
                for j in letter:
                    if tmpword[i]==j:continue
                    tmpword[i]=j
                    tmp=''.join(tmpword)
                    if tmp==endWord:
                        end=level
                    if (tmp in wordlist and (tmp not in dic or dic[tmp]&gt;=level+1)) or tmp==endWord:
                        if tmp not in self.father:
                            self.father[tmp]=set()
                        self.father[tmp].add(word)
```

                    if
mp not in dic and tmp!=endWord:

queue.append(tmp)

dic[tmp]=level+1

        self.buildPa
h([endWord])
        return self.r
s

    def buildPath(
elf,path0):
        path=path0[
:]
        if path[-1] n
t in self.father:return
        if not self.fa
her[path[-1]]:
            self.res.a
pend(path[::-1])
        for p in self.
ather[path[-1]]:
            path.app
nd(p)
            self.build
ath(path)
            path.pop(
1)
</code></pre>

<hr>
##151. Reverse Words in a String （反转）

难度：Medium

题意：
输入一个字符串，要求以单词为单位进行反转。

思路：
按照空格分割成列表，将列表反转后再用空格连接。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param s, a
string
    # @return a st

ing
    def reverseWo
ds(self, s):
        return ' '.joi
(s.split()[::-1])
</code></pre>

<hr>
##165. Compare Version Numbers

难度：Easy

题意：
比较两个版本号大小。

思路：
版本号大小规则与 float 大小规则不一样，版本号点后面的部分不是小数部分，而应该当作整数来对
。将版本号以第一个点分割成左边 left 和右边 right，左边大小直接决定了版本号的大小关系，若相
则递归比较右边（有可能出现多个点,右边也是一个完整的版本号）。这道 easy 题的通过率比很多 Ha
d 都低。

代码：
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
    # @param {str
ng} version1
    # @param {str
ng} version2
    # @return {int
ger}
    def compareV
rsion(self, version1, version2):
        left1,right1
self.splitVersion(version1)
        left2,right2
self.splitVersion(version2)
        if left1>l
ft2:return 1
        elif left1<
eft2:return -1
        elif left1==l
ft2 and not right1 and not right2:return 0
        else:
            return self
compareVersion(right1,right2)

    def splitVersio
(self,version):
        if not versio

:return 0,''
   i = version.f
nd('.')
   if i==-1:
   return int
version),''
   return int(ve
sion[:i]),version[i+1:]
</code></pre>

<hr>
<p>##214. Shortest Palindrome （回文）</p>

<p>难度：Hard</p>

<p>题意：
给定一个字符串 S，可以通过在字符串头部添加字符将 S 转换为回文串。返回通过这种转换方式能够
到的最短回文串。</p>

<p>思路：
求第一个字符开始的最长回文串，然后把剩下的部分逆序补到头部即可。因此这道题的难点在如何找
以第一个字符开始的最长回文串，而不超时。
这里推荐一种很巧妙的算法 mancher，mancher 在 O（n）时间复杂度内可以求出以每个字符为中
的回文串长度。算法巧妙之处在与，在字符中间插入了特殊符号，使奇和偶回文统一处理。是用辅助
组 P[i]记录以字符串 s[i]为中心的最长回文串可以向左或向右扩张的长度（包括自己）。记录当前最
回文串中心和覆盖范围，然后大家就画图观察吧，这个算法我也是画图弄了很久才学明白，然而还没
法用语言表达出来。</p>

<p>代码：</p>
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution(object):
   def shortestPal
ndrome(self, s):
   """
   :type s: str
   :rtype: str
   """
   pos = self.
anacher(s)
   return ''.join
[s[pos::][::-1],s])

   def manacher(
elf,s):
   s1 = list('#'.
oin(s))
   s1.insert(0,'*
)

   s1.append('
')
   p=[0]*len(s1

   mi,res=0,0
mi能覆盖右侧最远的回文串中心，res以第一个字符为开始的最大回文串
   for i in rang
(1,len(s1)-1):
   if i < m
+p[mi]:
   p[i] =
in(p[2*mi-i],mi+p[mi]-i)
   else:
   p[i] = 1
   while s1[
+p[i]]==s1[i-p[i]]:
   p[i]+=

   if mi+p[m
] < i+p[i]:
   mi = i
   if p[i]==i:
   res =
ax(res,i)
   return res
</code></pre>

<hr>
<p>##224. Basic Calculator （数学）</p>

<p>难度：Medium</p>

<p>题意：
实现一个基本的计算器，用于计算简单的字符串表示的算式。字符算式包括括号，+，-和非负整数和
格。</p>

<p>思路：
使用栈实现，一个栈记录数字(需要先把数字进行处理，去除空格转化为 int)，另一个栈记录符号，由
 + 和-是同级符号，只要不存在括号直接弹栈计算即可，注意考虑输入为各种可能。思路还是比较简
。</p>

<p>代码：</p>
<pre><code class="highlight-chroma"><span class="highlight
cl"> class Solution:
   # @param {str
ng} s
   # @return {int
ger}
<span class="highlight-line"><span class="highlight-cl">   def calculate(s

```
lf, s):
        result = 0
        nums = []
        ops = []
        if not s:
            return 0

        tmps = s.replace(' ', '')
        i=0
        while i &lt; len(tmps):
            #数字处理
            if tmps[i] not in('+','-','(',')'):
                numstr=[]
                while i &lt; len(tmps)  and tmps[i] not in ('+','-','(',')'):
                    numstr.append(tmps[i])
                    i += 1
                    ttt = ''.join(numstr)
                i -= 1
                nums.ppend(ttt)
            #+，-，

            elif tmps[i] in('+','-',')'):
                #若操栈中有非（的操作，出栈计算
                if ops and ops[-1] !='(':
                    numa = int(nums.pop())
                    numb = int(nums.pop())
                    op = ops.pop()
                    if op == '+':
                        result = num_b+num_a
                    else:
                        result = num_b-num_a
                    num.append(result)
                #若操栈中最后一个为)，新操作直接入栈
                if tmps[
```

```
] != ')':
                ops.ppend(tmps[i])
            else:
                ops.op()
        #（直接ops
        elif tmps[i] == '(':
            ops.append(tmps[i])
        i += 1

    #最后出栈
    if ops:
        num_a = nt(nums.pop())
        num_b = nt(nums.pop())
        op = ops.pop()
        if op == '+':
            result = num_b+num_a
        else:
            result = num_b-num_a
    else:
        result = int(nums.pop())
    return result
```


<hr>
##227. Basic Calculator II （数学）

难度：Medium

题意：
与 224 题意一致，不过这回少了括号，但是多了乘法和除法

思路：
还是使用两个栈来存储运算符和数字。运算符的优先级 <code>*=/ > +=-</code>，因此运算在入栈前，需要比较栈顶符号，若平级则弹出栈顶符号和对应的数字进行计算。若栈顶符号优先级低则直接入栈。

代码：

```
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">  class Solution:
</span></span><span class="highlight-line"><span class="highlight-cl">      # @param {str
ng} s
</span></span><span class="highlight-line"><span class="highlight-cl">      # @return {int
ger},
</span></span><span class="highlight-line"><span class="highlight-cl">      def calculate(s
lf, s):
</span></span><span class="highlight-line"><span class="highlight-cl">          s += '#'
</span></span><span class="highlight-line"><span class="highlight-cl">          num = 0
</span></span><span class="highlight-line"><span class="highlight-cl">          a = b = No
e
</span></span><span class="highlight-line"><span class="highlight-cl">          preop = op
= None
</span></span><span class="highlight-line"><span class="highlight-cl">          for c in s:
</span></span><span class="highlight-line"><span class="highlight-cl">              if c in ('+',
-','*','/','#'):
</span></span><span class="highlight-line"><span class="highlight-cl">                  if op is
None:
</span></span><span class="highlight-line"><span class="highlight-cl">                      a =
um
</span></span><span class="highlight-line"><span class="highlight-cl">                  elif op
n ('+','-'):
</span></span><span class="highlight-line"><span class="highlight-cl">                      if pr
op is None:
</span></span><span class="highlight-line"><span class="highlight-cl">                          b
= num
</span></span><span class="highlight-line"><span class="highlight-cl">                      else:
</span></span><span class="highlight-line"><span class="highlight-cl">                          a
 a + b if preop == '+' else a - b
</span></span><span class="highlight-line"><span class="highlight-cl">                      b
= num
</span></span><span class="highlight-line"><span class="highlight-cl">                      pre
p = op
</span></span><span class="highlight-line"><span class="highlight-cl">                  else:
</span></span><span class="highlight-line"><span class="highlight-cl">                      if pr
op is None:
</span></span><span class="highlight-line"><span class="highlight-cl">                          a
 a * num if op == '*' else a / num
</span></span><span class="highlight-line"><span class="highlight-cl">                      else:
</span></span><span class="highlight-line"><span class="highlight-cl">                          b
= b * num if op == '*' else b / num
</span></span><span class="highlight-line"><span class="highlight-cl">                  op = c
</span></span><span class="highlight-line"><span class="highlight-cl">                  num =
0
</span></span><span class="highlight-line"><span class="highlight-cl">              elif c != ' '

</span></span><span class="highlight-line"><span class="highlight-cl">                  num =
num * 10 + int(c)
</span></span><span class="highlight-line"><span class="highlight-cl">          if preop is
one:
</span></span><span class="highlight-line"><span class="highlight-cl">              return a
</span></span><span class="highlight-line"><span class="highlight-cl">          return a + b
f preop == '+' else a - b
```

</code></pre>

<hr>
##273. Integer to English Words （格式化）

难度：Medium
题意：
将数字转化为英文的表示方式。数字小于 2^31-1。例如：123 -> "One Hundred Twenty Three

思路：
思路比较简单，英文的数字表达方式是 3 位为一组，每 3 位的表达方式相同，最后再加入单位即可。意 3 位全为 0 和输入就是 0。注意英文拼写。
代码：

class Solution(object):
<pre><code class="highlight-chroma">one_nine=['0','One','Two','Three','Four','Five','Six','Seven','Eight','Nine']
ten_nineteen=['Te','Eleven','Twelve','Thirteen','Fourteen','Fifteen','Sixteen','Seventeen','Eighteen','Nineteen']
twenty_ninety=['0''0','Twenty','Thirty','Forty','Fifty','Sixty','Seventy','Eighty','Ninety']
unit=['','Thousand''Million','Billion','Trillion']

def numberToWods(self, num):
    """
    :type num: int
    :rtype: str
    """
    num,i,res = str(um),0,[]
    while True:
        temp = self.tree(int(num[-3:]))
        if temp:
            if i!=0:
                res.append(self.unit[i])
            res.appendtemp)
        if len(num)&l;=3:break
        num = num[:3]
        i+=1
    return 'Zero' if ot res else ' '.join(res[::-1])

def three(self,num:
    res = ""
    if num//100&gt

```
0:
</span></span><span class="highlight-line"><span class="highlight-cl">        res = ' '.join([
es,self.one_nine[num//100],'Hundred'])
</span></span><span class="highlight-line"><span class="highlight-cl">        num%=100
</span></span><span class="highlight-line"><span class="highlight-cl">    if 10&lt;=num&
t;=19:
</span></span><span class="highlight-line"><span class="highlight-cl">        res = ' '.join([
es,self.ten_nineteen[num-10]])
</span></span><span class="highlight-line"><span class="highlight-cl">    elif num&gt;=2
:
</span></span><span class="highlight-line"><span class="highlight-cl">        res = ' '.join([
es,self.twenty_ninety[num//10]])
</span></span><span class="highlight-line"><span class="highlight-cl">        num%=10
</span></span><span class="highlight-line"><span class="highlight-cl">    if 0&lt;num&lt;
0:
</span></span><span class="highlight-line"><span class="highlight-cl">        res = ' '.join([
es,self.one_nine[num]])
</span></span><span class="highlight-line"><span class="highlight-cl">    return res[1:]
</span></span></code></pre>
```