苏州大学计算机复试上机【字符串/数组类习题】【PAT和LeetCode习题】

PAT

1028 人口普查 (20分)

某城镇进行人口普查，得到了全体居民的生日。现请你写个程序，找出镇上最年长和最年轻的人。

这里确保每个输入的日期都是合法的，但不一定是合理的——假设已知镇上没有超过 200 岁的老人，而今天是 2014 年 9 月 6 日，所以超过 200 岁的生日和未出生的生日都是不合理的，应该被过滤掉。

输入格式：

输入在第一行给出正整数 *N*，取值在(0,105]；随后 *N* 行，每行给出 1 个人的姓名（由不超过 5 个英文字母组成的字符串）、以及按 `yyyy/mm/dd`（即年/月/日）格式给出的生日。题目保证最年长和最年轻的人没有并列。

输出格式：

在一行中顺序输出有效生日的个数、最年长人和最年轻人的姓名，其间以空格分隔。

输入样例：

```
5
John 2001/05/12
Tom 1814/09/06
Ann 2121/01/30
James 1814/09/05
Steve 1967/11/20
```

输出样例：

```
3 Tom John
```

代码：

```
# -*- coding: utf-8 -*-
def solve():
    n = int(input())
    count = 0
    res = []
#    datas = [('John', '2001/05/12'), ('Tom', '1814/09/06'), ('Ann',
'2121/01/30'), ('James', '1814/09/05'), ('Steve', '1967/11/20')]
    for i in range(n):
        elem = input().split()
        if '1814/09/06' <= elem[1] <= '2014/09/06':
            res.append(elem)
            count += 1
    res.sort(key=lambda x:x[1])

    if res:
        print(count, res[0][0], res[-1][0])
    else:
        print('0')

solve()
```

1029 旧键盘 (20分)

旧键盘上坏了几个键，于是在敲一段文字的时候，对应的字符就不会出现。现在给出应该输入的一段文字、以及实际被输入的文字，请你列出肯定坏掉的那些键。

输入格式：

输入在 2 行中分别给出应该输入的文字、以及实际被输入的文字。每段文字是不超过 80 个字符的串，由字母 A-Z（包括大、小写）、数字 0-9、以及下划线 `_`（代表空格）组成。题目保证 2 个字符串均非空。

输出格式：

按照发现顺序，在一行中输出坏掉的键。其中英文字母只输出大写，每个坏键只输出一次。题目保证至少有 1 个坏键。

输入样例：

```
7_This_is_a_test
_hs_s_a_es
```

输出样例：

```
7TI
```

代码：

```python
# -*- coding: utf-8 -*-

def solve():
    right = input().lower()
    wrong = input().lower()

    s1, s2 = set(right), set(wrong)
    out = list(s1 - s2)

    output = [[right.index(letter), letter] for letter in out]
    output.sort()

    olen = len(output)
    res = [str(output[i][1]) for i in range(olen)]

    print(''.join(res).upper())

solve()
```

1078 字符串压缩与解压 (20分)

文本压缩有很多种方法，这里我们只考虑最简单的一种：把由相同字符组成的一个连续的片段用这个字符和片段中含有这个字符的个数来表示。例如 `ccccc` 就用 `5c` 来表示。如果字符没有重复，就原样输出。例如 `aba` 压缩后仍然是 `aba`。

解压方法就是反过来，把形如 `5c` 这样的表示恢复为 `ccccc`。

本题需要你根据压缩或解压的要求，对给定字符串进行处理。这里我们简单地假设原始字符串是完全由英文字母和空格组成的非空字符串。

输入格式：

输入第一行给出一个字符，如果是 `C` 就表示下面的字符串需要被压缩；如果是 `D` 就表示下面的字符串需要被解压。第二行给出需要被压缩或解压的不超过 1000 个字符的字符串，以回车结尾。题目保证字符重复个数在整型范围内，且输出文件不超过 1MB。

输出格式：

根据要求压缩或解压字符串，并在一行中输出结果。

输入样例 1：

```
C
TTTTThhiiiis isssss a   tesssst CAaaa as
```

输出样例 1：

```
5T2h4is i5s a3 te4st CA3a as
```

输入样例 2：

```
D
5T2h4is i5s a3 te4st CA3a as1OZ
```

输出样例 2：

```
TTTTThhiiiis isssss a   tesssst CAaaa asZZZZZZZZZZ
```

代码：

```
# -*- coding: utf-8 -*-
import re

def solve():
    j = input()
    if j == 'C':      # 压缩
        data = input()
        data = [i+j for i,j in re.findall(r'([a-zA-Z\s])(\1*)', data)]  #
注意:r''不能少
        res = ''
        for elem in data:
            res += str(len(elem))+elem[0] if len(elem) > 1 else elem[0]
        print(res)
    else:             # 解压
        data = input()
        data = re.findall('(\d*)([a-zA-Z\s])', data)
        res = ''
        for length, elem in data:
            if length == '':
                res += elem
            else:
```

```
                res += int(length)*elem
        print(res)

solve()
```

1081 检查密码 (15分)

本题要求你帮助某网站的用户注册模块写一个密码合法性检查的小功能。该网站要求用户设置的密码必须由不少于6个字符组成，并且只能有英文字母、数字和小数点 `.`，还必须既有字母也有数字。

输入格式：

输入第一行给出一个正整数 N（≤ 100），随后 N 行，每行给出一个用户设置的密码，为不超过 80 个字符的非空字符串，以回车结束。

输出格式：

对每个用户的密码，在一行中输出系统反馈信息，分以下5种：

- 如果密码合法，输出`Your password is wan mei.`；
- 如果密码太短，不论合法与否，都输出`Your password is tai duan le.`；
- 如果密码长度合法，但存在不合法字符，则输出`Your password is tai luan le.`；
- 如果密码长度合法，但只有字母没有数字，则输出`Your password needs shu zi.`；
- 如果密码长度合法，但只有数字没有字母，则输出`Your password needs zi mu.`。

输入样例：

```
5
123s
zheshi.wodepw
1234.5678
WanMei23333
pass*word.6
```

代码：

```
# -*- coding: utf-8 -*-
import re

def solve():
    n = int(input())
    i = 0
```

```
    for i in range(n):
#         密码必须由不少于6个字符组成，并且只能有英文字母、数字和小数点 . ，还必须既有字母也有数字
        pwd = input()

        plen = len(pwd)
        sign = re.findall('[^0-9a-zA-Z.]', pwd)      # 存在不合法字符
        num = re.findall('[0-9]', pwd)               # 是否只有字母
        alpha = re.findall('[a-zA-Z]', pwd)          # 是否只有数字
        if plen < 6:
            print("Your password is tai duan le.")
        elif sign:
            print("Your password is tai luan le.")
        elif num == []:
            print("Your password needs shu zi.")
        elif alpha == []:
            print("Your password needs zi mu.")
        else:
            print("Your password is wan mei.")

solve()
```

1043 输出PATest (20分)

给定一个长度不超过 104 的、仅由英文字母构成的字符串。请将字符重新调整顺序，按 `PATestPATest....` 这样的顺序输出，并忽略其它字符。当然，六种字符的个数不一定是一样多的，若某种字符已经输出完，则余下的字符仍按 PATest 的顺序打印，直到所有字符都被输出。

输入格式：

输入在一行中给出一个长度不超过 104 的、仅由英文字母构成的非空字符串。

输出格式：

在一行中按题目要求输出排序后的字符串。题目保证输出非空。

输入样例：

```
redlesPayBestPATTopTeePHPereatitAPPT
```

输出样例：

```
PATestPATestPTetPTePePee
```

代码：

```
# -*- coding: utf-8 -*-
def solve():
    data = input()
    dlen = len(data)

    res = ''
    pattern = 'PATest'
    for i in range(dlen):
        for m in pattern:
            if data.find(m) != -1:
                data = data.replace(m, '', 1)
                res += m
    print(res)

solve()
```

1091 N-自守数 (15分)

如果某个数 *K* 的平方乘以 *N* 以后，结果的末尾几位数等于 *K*，那么就称这个数为“*N*-自守数”。例如 3×922=25392，而 25392 的末尾两位正好是 92，所以 92 是一个 3-自守数。

本题就请你编写程序判断一个给定的数字是否关于某个 *N* 是 *N*-自守数。

输入格式：

输入在第一行中给出正整数 *M*（≤20），随后一行给出 *M* 个待检测的、不超过 1000 的正整数。

输出格式：

对每个需要检测的数字，如果它是 *N*-自守数就在一行中输出最小的 *N* 和 *N****K*2 的值，以一个空格隔开；否则输出 `No`。注意题目保证 *N*<10。

输入样例：

```
3
92 5 233
```

输出样例：

```
3 25392
1 25
No
```

代码：

```
# -*- coding: utf-8 -*-
import math

def solve():
    M = int(input())
    data = list(map(int, input().split()))
    for K in data:
        for N in range(1, 10):
            t = N * K**2
            if t % 1000 == K or t % 100 == K or t % 10 == K:
                print(N, N * K**2)
                break
        else:
            print('No')

solve()
```

LeetCode

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。
```

代码：

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        if s == '':
            return 0
        if len(s) == 1:
            return 1

        def find_left(s, i):
            tmp_str = s[i]
            j = i - 1
            while j >= 0 and s[j] not in tmp_str:
                tmp_str += s[j]
                j -= 1
            return len(tmp_str)

        length = 0
        for i in range(0, len(s)):
            length = max(length, find_left(s, i))

        return length

s = Solution()
print(s.lengthOfLongestSubstring("pwwkew"))
print(s.lengthOfLongestSubstring("bbbbb"))
print(s.lengthOfLongestSubstring("abcabcbb"))
```

输出：

```
3
1
3
```

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"
```

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def longestPalindrome(self, s: str) -> str:
        length = len(s)
        result = ""
        for i in range(length):
            sum1 = ""
            sum2 = ""
            for str1 in s[i:]:
                sum1 = sum1 + str1
                sum2 = str1 + sum2
                if sum1 == sum2 and len(sum1) > len(result):
                    result = sum1
                else:
                    continue
        return result

s = Solution()
print(s.longestPalindrome('babad'))
print(s.longestPalindrome('cbbd'))
```

输出：

```
bab
bb
```

791. 自定义字符串排序

字符串`S`和 `T` 只包含小写字符。在`S`中，所有字符只会出现一次。

`S` 已经根据某种规则进行了排序。我们要根据`S`中的字符顺序对`T`进行排序。更具体地说，如果`S`中`x`在`y`之前出现，那么返回的字符串中`x`也应出现在`y`之前。

返回任意一种符合条件的字符串`T`。

```
示例:
输入:
S = "cba"
T = "abcd"
输出: "cbad"
解释:
S中出现了字符 "a", "b", "c", 所以 "a", "b", "c" 的顺序应该是 "c", "b", "a".
由于 "d" 没有在S中出现, 它可以放在T的任意位置. "dcba", "cdba", "cbda" 都是合法的输出。
```

代码：

```
#[('c', 1), ('b', 1), ('a', 1), ('f', 1), ('g', 1)]
#[('a', 1), ('b', 2), ('c', 1), ('d', 1)]
#['c', 'b', 'a', 'd']
#['c', 'b', 'b', 'a', 'd']
# cbbad
#{'c': 1, 'b': 2, 'a': 1, 'f': 0, 'g': 0}
# dcbba

class Solution:
    def customSortString(self, S: str, T: str) -> str:
        import collections
        cs = collections.Counter(S)
        print(list(cs.items()))
        ct = collections.Counter(T)
        print(list(ct.items()))

        print(list((cs + ct - cs)))
```

```
        print(list((cs + ct - cs).elements()))

        # cs + ct - cs
        return ''.join(list((cs + ct - cs).elements()))

    def customSortString2(self, S, T):
        d = {c:0 for c in S}
        ans = ''
        for c in T:
            if c in d:
                d[c] += 1
            else:
                ans += c

        print(d)
        for c in d:
            ans += c * d[c]

        return ans

s = Solution()
print(s.customSortString('cbafg', 'abcdb'))
print(s.customSortString2('cbafg', 'abcdb'))
```

804. 唯一摩尔斯密码词

国际摩尔斯密码定义一种标准编码方式，将每个字母对应于一个由一系列点和短线组成的字符串， 比如: `"a"` 对应 `".-"`, `"b"` 对应 `"-..."`, `"c"` 对应 `"-.-."`, 等等。

为了方便，所有26个英文字母对应摩尔斯密码表如下：

```
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-
",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--
","-..-","-.--","--.."]
```

给定一个单词列表，每个单词可以写成每个字母对应摩尔斯密码的组合。例如，"cab" 可以写成 "-.-..--..."，(即 "-.-." + "-..." + ".-"字符串的结合)。我们将这样一个连接过程称作单词翻译。

返回我们可以获得所有词不同单词翻译的数量。

```
例如:
输入: words = ["gin", "zen", "gig", "msg"]
```

```
输出: 2
解释:
各单词翻译如下:
"gin" -> "--...-."
"zen" -> "--...-."
"gig" -> "--...--."
"msg" -> "--...--."

共有 2 种不同翻译, "--...-." 和 "--...--.".

注意:

- 单词列表words 的长度不会超过 100。
- 每个单词 words[i]的长度范围为 [1, 12]。
- 每个单词 words[i]只包含小写字母。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def uniqueMorseRepresentations(self, words) -> int:
        import string
        letters = list(string.ascii_lowercase)
        table = [".-","-...","-.-.","-..",".","..-.","-
-.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-
",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]
        wlen = len(words)
        T = []
        for i in range(wlen):
            str2 = ''
            wl = len(words[i])          # 求单词的编码
            for j in range(wl):
                k = letters.index(words[i][j])
                str2 += table[k]
            if str2 not in T:
                T.append(str2)

        return len(T)

s = Solution()
print(s.uniqueMorseRepresentations(["gin", "zen", "gig", "msg"]))
```

856. 括号的分数

给定一个平衡括号字符串 `S`，按下述规则计算该字符串的分数：

- `()` 得 1 分。
- `AB` 得 `A + B` 分，其中 A 和 B 是平衡括号字符串。
- `(A)` 得 `2 * A` 分，其中 A 是平衡括号字符串。

示例 1：

```
输入： "()"
输出： 1
```

示例 2：

```
输入： "(())"
输出： 2
```

示例 3：

```
输入： "()()"
输出： 2
```

示例 4：

```
输入： "(()(()))"
输出： 6
```

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def scoreOfParentheses(self, S: str) -> int:
        stack = []
        res = 0
        for i in S:
            if i == '(':
                stack.append(res)
                res = 0
            elif i == ')':
```

```
                p = stack.pop()
                res = p + 1 if res == 0 else p + 2 * res
            else:
                res += int(i)
        return res

s = Solution()
print(s.scoreOfParentheses('(())'))
print(s.scoreOfParentheses("(()(()))"))
print(s.scoreOfParentheses("()"))
```

890. 查找和替换模式

你有一个单词列表 `words` 和一个模式 `pattern`，你想知道 `words` 中的哪些单词与模式匹配。

如果存在字母的排列 `p` ，使得将模式中的每个字母 `x` 替换为 `p(x)` 之后，我们就得到了所需的单词，那么单词与模式是匹配的。

（回想一下，字母的排列是从字母到字母的双射：每个字母映射到另一个字母，没有两个字母映射到同一个字母。）

返回 `words` 中与给定模式匹配的单词列表。

你可以按任何顺序返回答案。

代码:

```
# -*- coding: utf-8 -*-
#    输入：words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"
#    输出：["mee","aqq"]
#    解释：
#    "mee" 与模式匹配，因为存在排列 {a -> m, b -> e, ...}。
#    "ccc" 与模式不匹配，因为 {a -> c, b -> c, ...} 不是排列。
#    因为 a 和 b 映射到同一个字母。
class Solution:
    def findAndReplacePattern(self, words, pattern: str):
        # 是否有模式匹配
        def check(word, pattern):
            if len(word) != len(pattern):
                return False
            for i in range(len(word)):
                if word.index(word[i]) != pattern.index(pattern[i]):
```

```
                    return False
            return True
        res = []
        for word in words:
             if check(word, pattern):
               res.append(word)
        print(res)
        return res

s = Solution()
s.findAndReplacePattern(["abc","deq","mee","aqq","dkd","ccc"], "abb")
```

面试题 08.09. 括号

括号。设计一种算法，打印n对括号的所有合法的（例如，开闭一一对应）组合。

说明：解集不能包含重复的子集。

例如，给出 n = 3，生成结果为：

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def generateParenthesis(self, n: int):
        res = []
        state = ''

        def dsp(state, p, q):
            if p > q:
                return
            if q == 0:
                res.append(state)
```

```
            if p > 0:          # p == q时候，则递归搜索
                dsp(state + '(', p - 1, q)
            if q > 0:
                dsp(state + ')', p, q - 1)

        dsp(state, n, n)
        return res

s = Solution()
print(s.generateParenthesis(3))
print(s.generateParenthesis(4))
```

输出：

```
['((()))', '(()())', '(())()', '()(())', '()()()']
['(((())))', '((()()))', '((())())', '((()))()', '(()(()))', '(()()())',
'(()())()', '(())(())', '(())()()', '()((()))', '()(()())', '()(())()',
'()()(())', '()()()()']
```

929. 独特的电子邮件地址

每封电子邮件都由一个本地名称和一个域名组成，以 @ 符号分隔。

例如，在 `alice@leetcode.com`中， `alice` 是本地名称，而 `leetcode.com` 是域名。

除了小写字母，这些电子邮件还可能包含 `'.'` 或 `'+'`。

如果在电子邮件地址的**本地名称**部分中的某些字符之间添加句点（ `'.'` ），则发往那里的邮件将会转发到本地名称中没有点的同一地址。例如，`"alice.z@leetcode.com”` 和 `“alicez@leetcode.com”` 会转发到同一电子邮件地址。（请注意，此规则不适用于域名。）

如果在**本地名称**中添加加号（ `'+'` ），则会忽略第一个加号后面的所有内容。这允许过滤某些电子邮件，例如 `m.y+name@email.com` 将转发到 `my@email.com`。（同样，此规则不适用于域名。）

可以同时使用这两个规则。

给定电子邮件列表 `emails`，我们会向列表中的每个地址发送一封电子邮件。实际收到邮件的不同地址有多少？

示例：

```
输入：
["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","tes
temail+david@lee.tcode.com"]
输出：2
解释：实际收到邮件的是 "testemail@leetcode.com" 和
"testemail@lee.tcode.com"。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def numUniqueEmails(self, emails):
        import re
        eset = set()
        for email in emails:
            s = email.split('@')
#            print('email.split(\'@\')', s)
            s = s[0].replace('.', '') + '@' + s[1]
#            print('\nreplace: ', s, '\n')
            # 替换 正则表达式匹配内容 为 子串‘@’
            # \+ 代表是 + 符号，遇到@ 停止匹配，期间的字符都删除
            s = re.sub(r'\+.*@', '@', s)
            eset.add(s)

#        print(eset)
        return len(eset)

s = Solution()
s.numUniqueEmails(["test.email+alex@leetcode.com","test.e.mail+bob.cathy@
leetcode.com","testemail+david@lee.tcode.com"])
```

1170. 比较字符串最小字母出现频次

我们来定义一个函数 `f(s)`，其中传入参数 `s` 是一个非空字符串；该函数的功能是统计 `s` 中（按字典序比较）最小字母的出现频次。

例如，若 `s = "dcce"`，那么 `f(s) = 2`，因为最小的字母是 `"c"`，它出现了 2 次。

现在，给你两个字符串数组待查表 `queries` 和词汇表 `words`，请你返回一个整数数组 `answer` 作为答案，其中每个 `answer[i]` 是满足 `f(queries[i])` < `f(W)` 的词的数目，`W` 是词汇表 `words` 中的词。

示例 1：

```
输入：queries = ["cbd"], words = ["zaaaz"]
输出：[1]
解释：查询 f("cbd") = 1，而 f("zaaaz") = 3 所以 f("cbd") < f("zaaaz")。
输入：queries = ["bbb","cc"], words = ["a","aa","aaa","aaaa"]
输出：[1,2]
解释：第一个查询 f("bbb") < f("aaaa")，第二个查询 f("aaa") 和 f("aaaa") 都 > f("cc")。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def numSmallerByFrequency(self, queries, words):
        def check(s):
            return s.count(min(s))
        res = []
        # 先存储一遍words中的最小值，以免重复计算
        words_cnt = [check(word) for word in words]
        for item in queries:
            # 避免重复计算
            f_item = check(item)
            res.append(len([1 for cnt in words_cnt if f_item < cnt]))
#         print(res)
        return res

s = Solution()
s.numSmallerByFrequency(queries = ["cbd"], words = ["zaaaz"])
s.numSmallerByFrequency(queries = ["bbb","cc"], words =
["a","aa","aaa","aaaa"])
s.numSmallerByFrequency(["bba","abaaaaaa","aaaaaa","bbabbabaab","aba","aa
","baab","bbbbbb","aab","bbabbaabb"],
["aaabbb","aab","babbab","babbbb","b","bbbbbbbbab","a","bbbbbbbbbb","baaa
bbaab","aa"])
```

1071. 字符串的最大公因子

对于字符串 `S` 和 `T`，只有在 `S = T + ... + T`（`T` 与自身连接 1 次或多次）时，我们才认定“`T` 能除尽 `S`”。

返回最长字符串 `X`，要求满足 `X` 能除尽 `str1` 且 `X` 能除尽 `str2`。

示例 1：

```
输入：str1 = "ABCABC", str2 = "ABC"
输出："ABC"
```

示例 2：

```
输入：str1 = "ABABAB", str2 = "ABAB"
输出："AB"
```

示例 3：

```
输入：str1 = "LEET", str2 = "CODE"
输出：""
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def gcdOfStrings(self, str1: str, str2: str) -> str:
        import math

        i = j = 0
        m, n = len(str1), len(str2)
        while i < m or j < n:
            if str1[i % m] != str2[j % n]:
                return ''
            i += 1
            j += 1

        return str1[:math.gcd(m, n)]

s = Solution()
print(s.gcdOfStrings(str1 = "ABCABC", str2 = "ABC"))
print(s.gcdOfStrings(str1 = "ABABAB", str2 = "ABAB"))
print(s.gcdOfStrings(str1 = "LEET", str2 = "CODE"))
```

1221. 分割平衡字符串

在一个「平衡字符串」中，'L' 和 'R' 字符的数量是相同的。

给出一个平衡字符串 `s`，请你将它分割成尽可能多的平衡字符串。

返回可以通过分割得到的平衡字符串的最大数量。

示例 1：

```
输入：s = "RLRRLLRLRL"
输出：4
解释：s 可以分割为 "RL", "RRLL", "RL", "RL", 每个子字符串中都包含相同数量的 'L' 和 'R'。
```

示例 2：

```
输入：s = "RLLLLRRRLR"
输出：3
解释：s 可以分割为 "RL", "LLLRRR", "LR", 每个子字符串中都包含相同数量的 'L' 和 'R'。
```

示例 3：

```
输入：s = "LLLLRRRR"
输出：1
解释：s 只能保持原样 "LLLLRRRR".

提示：
- 1 <= s.length <= 1000
- s[i] = 'L' 或 'R'
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def balancedStringSplit(self, s: str) -> int:
        ans = 0
        n = 0
        for e in s:
            ans = ans - 1 if e == 'L' else ans + 1
            if ans == 0:      # 模拟栈
                n += 1
        return n
```

```
s = Solution()
print(s.balancedStringSplit("RLLLLRRRLR"))
print(s.balancedStringSplit("RLRRLLRLRL"))
```

输出：

```
3
4
```

1324. 竖直打印单词

给你一个字符串 `s`。请你按照单词在 `s` 中的出现顺序将它们全部竖直返回。 单词应该以字符串列表的形式返回，必要时用空格补位，但输出尾部的空格需要删除（不允许尾随空格）。 每个单词只能放在一列上，每一列中也只能有一个单词。

示例 1：

```
输入：s = "HOW ARE YOU"
输出：["HAY","ORO","WEU"]
解释：每个单词都应该竖直打印。
 "HAY"
 "ORO"
 "WEU"
```

示例 2：

```
输入：s = "TO BE OR NOT TO BE"
输出：["TBONTB","OEROOE","   T"]
解释：题目允许使用空格补位，但不允许输出末尾出现空格。
"TBONTB"
"OEROOE"
"   T"
```

示例 3：

```
输入：s = "CONTEST IS COMING"
输出：["CIC","OSO","N M","T I","E N","S G","T"]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def printVertically(self, s: str):
        s = s.split(' ')
        max_1 = 0
        for w in s:
            max_1 = max(max_1, len(w))

        r = [[] for i in range(max_1)]

        for w in s:
            for i in range(max_1):
                if i < len(w):
                    # 第i个列表，存放word的第i个单词
                    r[i].append(w[i])
                else:
                    r[i].append(' ')
        res = []
        # 去除末尾空格
        for x in r:
            res.append(''.join(x).rstrip())
        return res

s = Solution()
print(s.printVertically(s = "TO BE OR NOT TO BE"))
```

1332. 删除回文子序列

给你一个字符串 `s`，它仅由字母 `'a'` 和 `'b'` 组成。每一次删除操作都可以从 `s` 中删除一个回文 **子序列**。

返回删除给定字符串中所有字符（字符串为空）的最小删除次数。

「子序列」定义：如果一个字符串可以通过删除原字符串某些字符而不改变原字符顺序得到，那么这个字符串就是原字符串的一个子序列。

示例 1：

```
输入：s = "ababa"
输出：1
解释：字符串本身就是回文序列，只需要删除一次。
```

```
输入：s = "abb"
输出：2
解释："abb" -> "bb" -> "".
先删除回文子序列 "a"，然后再删除 "bb"。

输入：s = "baabb"
输出：2
解释："baabb" -> "b" -> "".
先删除回文子序列 "baab"，然后再删除 "b"。

输入：s = ""
输出：0
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def removePalindromeSub(self, s: str) -> int:
        def isPalid(word):
            wlen = len(word)
            for i in range(wlen):
                if word[i] != word[wlen - i - 1]:
                    return False
            return True
        sli = s[::]
        res = 0
        for e in sli:
            if isPalid(s):
                res += 1
                return res
            else:
                res += 1
                s = s.replace(e, '')
        return 0

s = Solution()
print(s.removePalindromeSub('ababa'))
print(s.removePalindromeSub('abb'))
print(s.removePalindromeSub('baabb'))
print(s.removePalindromeSub(''))
```

1370. 上升下降字符串

给你一个字符串 `s` ，请你根据下面的算法重新构造字符串：

1. 从 `s` 中选出 **最小** 的字符，将它 **接在** 结果字符串的后面。
2. 从 `s` 剩余字符中选出 **最小** 的字符，且该字符比上一个添加的字符大，将它 **接在** 结果字符串后面。
3. 重复步骤 2 ，直到你没法从 `s` 中选择字符。
4. 从 `s` 中选出 **最大** 的字符，将它 **接在** 结果字符串的后面。
5. 从 `s` 剩余字符中选出 **最大** 的字符，且该字符比上一个添加的字符小，将它 **接在** 结果字符串后面。
6. 重复步骤 5 ，直到你没法从 `s` 中选择字符。
7. 重复步骤 1 到 6 ，直到 `s` 中所有字符都已经被选过。

在任何一步中，如果最小或者最大字符不止一个 ，你可以选择其中任意一个，并将其添加到结果字符串。

请你返回将 `s` 中字符重新排序后的 **结果字符串** 。

示例 1：

```
输入：s = "aaaabbbbcccc"
输出："abccbaabccba"
解释：第一轮的步骤 1，2，3 后，结果字符串为 result = "abc"
第一轮的步骤 4，5，6 后，结果字符串为 result = "abccba"
第一轮结束，现在 s = "aabbcc" ，我们再次回到步骤 1
第二轮的步骤 1，2，3 后，结果字符串为 result = "abccbaabc"
第二轮的步骤 4，5，6 后，结果字符串为 result = "abccbaabccba"

1.输入：s = "rat"
  输出："art"
解释：单词 "rat" 在上述算法重排序以后变成 "art"
2.输入：s = "leetcode"
  输出："cdelotee"
3.输入：s = "ggggggg"
  输出："ggggggg"
```

代码：

```
# -*- coding: utf-8 -*-

class Solution:
```

```
    def sortString(self, s: str) -> str:
        if not s:
            return ''

        s = list(s)
        res = []
        while s:                # len(s) 次循环
            c_list = list(set(s))
            c_list.sort()
            for i in c_list:
                res.append(i)
                s.remove(i)             # 从原来字符里删除

            c_list = list(set(s))
            c_list.sort(reverse=True)   # 倒序
            for i in c_list:
                res.append(i)
                s.remove(i)

        return ''.join(res)

s = Solution()
print(s.sortString("aaaabbbbcccc"))
print(s.sortString("rat"))
print(s.sortString("leetcode"))
print(s.sortString("ggggggg"))
```

1309. 解码字母到整数映射

给你一个字符串 `s`，它由数字（`'0'` - `'9'`）和 `'#'` 组成。我们希望按下述规则将 `s` 映射为一些小写英文字符：

- 字符（`'a'` - `'i'`）分别用（`'1'` - `'9'`）表示。
- 字符（`'j'` - `'z'`）分别用（`'10#'` - `'26#'`）表示。

返回映射之后形成的新字符串。

题目数据保证映射始终唯一。

示例 1：

```
输入：s = "10#11#12"
输出："jkab"
解释："j" -> "10#" , "k" -> "11#" , "a" -> "1" , "b" -> "2".
```

示例 2：

```
输入：s = "1326#"
输出："acz"
```

示例 3：

```
输入：s = "25#"
输出："y"
```

示例 4：

```
输入：s = "12345678910#11#12#13#14#15#16#17#18#19#20#21#22#23#24#25#26#"
输出："abcdefghijklmnopqrstuvwxyz"
```

提示：

- `1 <= s.length <= 1000`；`s[i]` 只包含数字（`'0'`-`'9'`）和 `'#'` 字符；`s` 是映射始终存在的有效字符串。

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def freqAlphabets(self, s: str) -> str:
        import string
        import re
        words = list(' ' + string.ascii_lowercase)

        li = s.split('#')
        res = ''                        # 结果
        wlen = len(li)
        for i in range(0, wlen- 1):
            w = li[i]                   # '12345610'
            length = len(w)
            if length <= 2:
                w = int(w)
```

```
                res += str(words[w])
            else:
                front = w[:-2]
                beh = int(w[-2:])
                for e in front:
                    res += str(words[int(e)])
                res += str(words[int(beh)])
        end = li[-1]
        if end != '':
            for e in end:
                w = int(e)
                res += str(words[w])
        return res

s = Solution()
s.freqAlphabets("10#11#12")
s.freqAlphabets("1326#")
s.freqAlphabets("25#")
```

1313. 解压缩编码列表

给你一个以行程长度编码压缩的整数列表 `nums` 。

考虑每对相邻的两个元素 `freq, val] = [nums[2*i], nums[2*i+1]]` （其中 `i >= 0` ），每一对都表示解压后子列表中有 `freq` 个值为 `val` 的元素，你需要从左到右连接所有子列表以生成解压后的列表。

请你返回解压后的列表。

示例：

```
输入：nums = [1,2,3,4]
输出：[2,4,4,4]
解释：第一对 [1,2] 代表着 2 的出现频次为 1，所以生成数组 [2]。
第二对 [3,4] 代表着 4 的出现频次为 3，所以生成数组 [4,4,4]。
最后将它们串联到一起 [2] + [4,4,4] = [2,4,4,4]。

输入：nums = [1,1,2,3]
输出：[1,3,3]
```

提示： `2 <= nums.length <= 100` ； `nums.length % 2 == 0` ； `1 <= nums[i] <= 100`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def decompressRLElist(self, nums):
        nlen = len(nums)
        points = [(nums[i], nums[i+1]) for i in range(0, nlen, 2)]
        res = []
        for e in points:
            freq = e[0]
            for i in range(freq):
                res.append(e[1])
        return res

s = Solution()
print(s.decompressRLElist([1, 2, 3, 4]))
```

1408. 数组中的字符串匹配

给你一个字符串数组 `words` ，数组中的每个字符串都可以看作是一个单词。请你按 **任意** 顺序返回 `words` 中是其他单词的子字符串的所有单词。

如果你可以删除 `words[j]` 最左侧和/或最右侧的若干字符得到 `word[i]` ，那么字符串 `words[i]` 就是 `words[j]` 的一个子字符串。

示例 1：

```
输入：words = ["mass","as","hero","superhero"]
输出：["as","hero"]
解释："as" 是 "mass" 的子字符串，"hero" 是 "superhero" 的子字符串。
["hero","as"] 也是有效的答案。
```

示例 2：

```
输入：words = ["leetcode","et","code"]
输出：["et","code"]
解释："et" 和 "code" 都是 "leetcode" 的子字符串。
```

示例 3：

```
输入：words = ["blue","green","bu"]
输出：[]
```

提示：`1 <= words.length <= 100`；`1 <= words[i].length <= 30`；`words[i]` 仅包含小写英文字母。题目数据 **保证** 每个 `words[i]` 都是独一无二的。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def stringMatching(self, words):
        ans = []
        wlen = len(words)
        for i in range(wlen):
            for j in range(wlen):
                ss = words[i]
                subs = words[j]
                if ss != subs and ss.find(subs) != -1:
                    ans.append(subs)
        return list(set(ans))

s = Solution()
print(s.stringMatching(words = ["mass","as","hero","superhero"]))
print(s.stringMatching(["leetcoder","leetcode","od","hamlet","am"]))
```

788. 旋转数字

我们称一个数 X 为好数, 如果它的每位数字逐个地被旋转 180 度后，我们仍可以得到一个有效的，且和 X 不同的数。要求每位数字都要被旋转。

如果一个数的每位数字被旋转以后仍然还是一个数字， 则这个数是有效的。0, 1, 和 8 被旋转后仍然是它们自己；2 和 5 可以互相旋转成对方（在这种情况下，它们以不同的方向旋转，换句话说，2 和 5 互为镜像）；6 和 9 同理，除了这些以外其他的数字旋转以后都不再是有效的数字。

现在我们有一个正整数 `N`, 计算从 `1` 到 `N` 中有多少个数 X 是好数？

示例：

```
输入: 10
输出: 4
解释:
在[1, 10]中有四个好数： 2, 5, 6, 9。
注意 1 和 10 不是好数，因为他们在旋转之后不变。
```

提示： N 的取值范围是 `[1, 10000]`。

```
# -*- coding: utf-8 -*-
class Solution:
    def rotatedDigits(self, N: int) -> int:
        ans = 0

        for i in range(1, N+1):
            flag = False
            n = i
            while n:
                x = n % 10
                if x in [3, 4, 7]:
                    break
                elif x in [2, 5, 6, 9]:
                    flag = True
                n = n // 10
            if flag and n == 0:
                ans = ans + 1

        return ans

s = Solution()
print(s.rotatedDigits(10))
```

面试题 01.07. 旋转矩阵

给你一幅由 `N × N` 矩阵表示的图像，其中每个像素的大小为 4 字节。请你设计一种算法，将图像旋转 90 度。

不占用额外内存空间能否做到？

示例 1:

```
给定 matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],

原地旋转输入矩阵，使其变为:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def rotate(self, matrix) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        N = len(matrix)

        #  matrix 顺时针转 90 度就是矩阵先上下翻转，后沿对角线翻转。
        matrix[:] = matrix[::-1]    # 先上下翻转
        for i in range(N):
            for j in range(i):
                # 沿对角线翻转
                matrix[j][i], matrix[i][j] = matrix[i][j], matrix[j][i]
        return matrix

s = Solution()
print(s.rotate(matrix = [[1,2,3],
                         [4,5,6],
                         [7,8,9]]))
```

78. 子集

给定一组**不含重复元素**的整数数组 *nums*，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例:

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

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def subsets(self, nums):
        res = [[]]

        for i in nums:
            print(i)
            res = res + [[i] + num for num in res]
        return res

s = Solution()
print(s.subsets([1,2,3]))
```

13. 罗马数字转整数

罗马数字包含以下七种字符: `I`， `V`， `X`， `L`，`C`，`D` 和 `M`。

字符	数值
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

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

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

- `I` 可以放在 `V` (5) 和 `X` (10) 的左边，来表示 4 和 9。
- `X` 可以放在 `L` (50) 和 `C` (100) 的左边，来表示 40 和 90。
- `C` 可以放在 `D` (500) 和 `M` (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。

示例 1:

```
输入: "III"
输出: 3
```

示例 2:

```
输入: "IV"
输出: 4
```

示例 3:

```
输入: "IX"
输出: 9
```

示例 4:

```
输入: "LVIII"
输出: 58
解释: L = 50, V= 5, III = 3.
```

示例 5:

```
输入: "MCMXCIV"
输出: 1994
解释: M = 1000, CM = 900, XC = 90, IV = 4.
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def romanToInt(self, s: str) -> int:
        dic = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}

        slen = len(s)
        ans = 0
        i = 0
        while i < slen:
            if i < slen - 1:
               if dic[s[i]] < dic[s[i+1]]:
                   ans += (dic[s[i+1]] - dic[s[i]])
                   i = i + 1
               else:
                   ans += dic[s[i]]
            else:
                ans += dic[s[i]]
            i = i + 1

        return ans

s = Solution()
print(s.romanToInt("III"))
print(s.romanToInt('IV'))
print(s.romanToInt('IX'))
print(s.romanToInt('LVIII'))
print(s.romanToInt("MCMXCIV"))
```

12. 整数转罗马数字

罗马数字包含以下七种字符： `I`， `V`， `X`， `L`，`C`，`D` 和 `M`。

字符	数值
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

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

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

- **`I` 可以放在 `V` (5) 和 `X` (10) 的左边，来表示 4 和 9。**
- **`X` 可以放在 `L` (50) 和 `C` (100) 的左边，来表示 40 和 90。**
- **`C` 可以放在 `D` (500) 和 `M` (1000) 的左边，来表示 400 和 900。**

给定一个整数，将其转为罗马数字。输入确保在 1 到 3999 的范围内。

示例 1:

```
输入: 3
输出: "III"
```

示例 2:

```
输入: 4
输出: "IV"
```

示例 3:

```
输入: 9
输出: "IX"
```

示例 4:

```
输入: 58
输出: "LVIII"
解释: L = 50, V = 5, III = 3.
```

示例 5:

```
输入: 1994
输出: "MCMXCIV"
解释: M = 1000, CM = 900, XC = 90, IV = 4.
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def intToRoman(self, num: int) -> str:
        hashmap = {1000:'M', 900:'CM', 500:'D', 400:'CD', 100:'C',
                   90:'XC', 50:'L', 40:'XL',
                   10:'X', 9:'IX', 5:'V' , 4:'IV', 1:'I'}
        ans = ''
        for key in hashmap:
            count = num // key
            if count != 0:
                ans += hashmap[key] * count
                num %= key
        return ans

s = Solution()
print(s.intToRoman(3))
print(s.intToRoman(4))
print(s.intToRoman(9))
print(s.intToRoman(58))
print(s.intToRoman(1994))
```

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 ]
]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def generateMatrix(self, n: int):
        matrix = [[0]*n for i in range(n)]

        num, tar = 1, n*n
        left, right = 0, n-1
        top, bottom = 0, n-1
        while num <= tar:
            # 左-->右
            for i in range(left, right + 1):
                matrix[top][i] = num
                num = num + 1
            top = top + 1
            # 上-->下
            for i in range(top, bottom + 1):
                matrix[i][right] = num
                num = num + 1
            right = right - 1
            # 右-->左
            for i in range(right, left - 1, -1):
                matrix[bottom][i] = num
                num = num + 1
            bottom = bottom - 1
            # 下-->上
            for i in range(bottom, top - 1, -1):
                matrix[i][left] = num
                num = num + 1
            left = left + 1

        return matrix
```

```
s = Solution()
print(s.generateMatrix(3))
```

1299. 将每个元素替换为右侧最大元素

给你一个数组 `arr` ，请你将每个元素用它右边最大的元素替换，如果是最后一个元素，用 `-1` 替换。

完成所有替换操作后，请你返回这个数组。

示例：

```
输入：arr = [17,18,5,4,6,1]
输出：[18,6,6,6,1,-1]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def replaceElements(self, arr):
        alen = len(arr)
        ans = [0]*alen
        ans[alen - 1] = -1
        # [alen-2, 0] 逆序
        for i in range(alen - 2, -1, -1):
#             print('ans[i+1]:', ans[i+1], ' arr[i+1]:', arr[i + 1])
            ans[i] = max(ans[i + 1], arr[i + 1])
        return ans

s = Solution()
print(s.replaceElements(arr = [17,18,5,4,6,1]))
```

1266. 访问所有点的最小时间

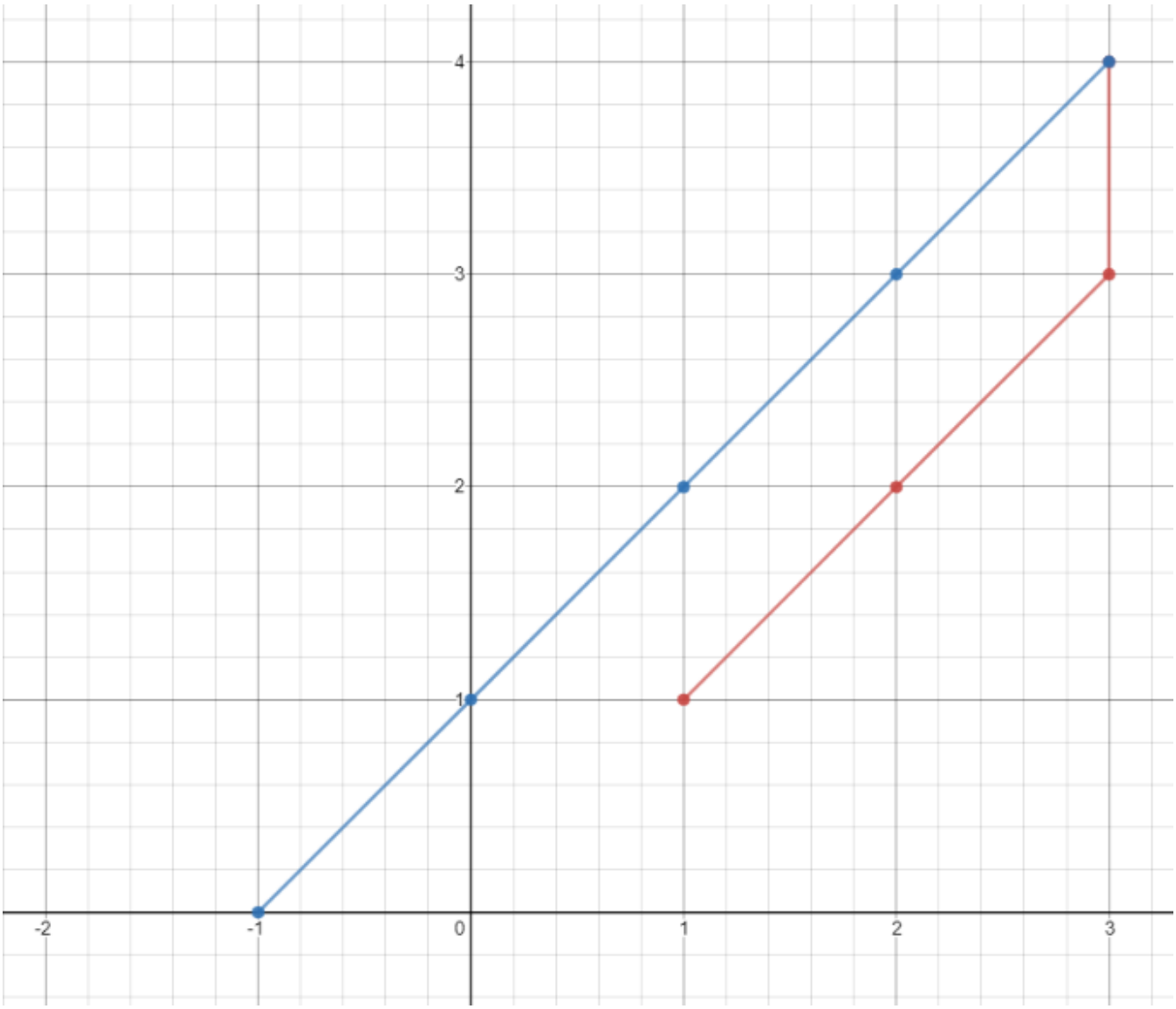
平面上有 `n` 个点，点的位置用整数坐标表示 `points[i] = [xi, yi]`。请你计算访问所有这些点需要的最小时间（以秒为单位）。

你可以按照下面的规则在平面上移动：

- 每一秒沿水平或者竖直方向移动一个单位长度，或者跨过对角线（可以看作在一秒内向水平和竖直方向各移动一个单位长度）。

- 必须按照数组中出现的顺序来访问这些点。

示例 1：




```
输入：points = [[1,1],[3,4],[-1,0]]
输出：7
解释：一条最佳的访问路径是： [1,1] -> [2,2] -> [3,3] -> [3,4] -> [2,3] ->
[1,2] -> [0,1] -> [-1,0]
从 [1,1] 到 [3,4] 需要 3 秒
从 [3,4] 到 [-1,0] 需要 4 秒
一共需要 7 秒
```

示例 2：

```
输入：points = [[3,2],[-2,2]]
输出：5
```

提示： `points.length == n` ; `1 <= n <= 100` ; `points[i].length == 2` ; `-1000 <= points[i][0], points[i][1] <= 1000`

代码：

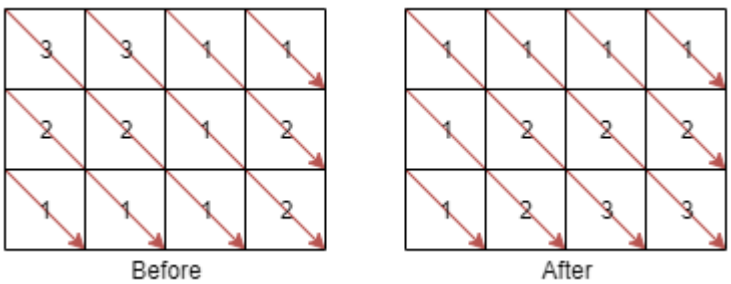
```
# -*- coding: utf-8 -*-
class Solution:
    def minTimeToVisitAllPoints(self, points) -> int:
        x0, x1 = points[0][0], points[0][1]
        ans = 0
        plen = len(points)
        for i in range(1, plen):
            y0, y1 = points[i][0], points[i][1]
            ans += max(abs(x0 - y0), abs(x1 - y1))
            x0, x1 = y0, y1
        return ans

s = Solution()
print(s.minTimeToVisitAllPoints(points = [[1,1],[3,4],[-1,0]]))
print(s.minTimeToVisitAllPoints(points = [[3,2],[-2,2]]))
```

1329. 将矩阵按对角线排序 [数组, 遍历对角线]

给你一个 `m * n` 的整数矩阵 `mat` ，请你将同一条对角线上的元素（从左上到右下）按升序排序后，返回排好序的矩阵。

示例 1：




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

提示： `m == mat.length` ; `n == mat[i].length` ; `1 <= m, n <= 100` ; `1 <= mat[i][j] <= 100`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def diagonalSort(self, mat):
        import collections
        import itertools
        row = len(mat)
        col = len(mat[0])
        # 初始化字典值为 []
        dic = collections.defaultdict(list)
        for i, j in itertools.product(range(row), range(col)):
            print(i, j)
            dic[i - j].append(mat[i][j])
        print(dic)
        dic = {k: iter(sorted(v)) for k, v in dic.items()}
        print(dic)
        for i, j in itertools.product(range(row), range(col)):
            mat[i][j] = next(dic[i - j])
        return mat

s = Solution()
print(s.diagonalSort(mat = [[3,3,1,1], [2,2,1,2], [1,1,1,2]]))
```