0.字符串总结.md

- 字符相关函数使用
  - 8 字符串转换整数 (atoi)
  - 58 最后一个单词的长度
  - 151 翻转字符串里的单词
  - 434 字符串中的单词数
  - 500 键盘行
  - 520 检测大写字母
  - 541 反转字符串 II
  - 557 反转字符串中的单词 III
  - 709 转换成小写字母
  - 744 寻找比目标字母大的最小字母
- 字符串检索
  - 28 实现 strStr()
  - 796 旋转字符串
- ord
  - 804 唯一摩尔斯密码词
  - 806 写字符串需要的行数
- 注意标点
  - 819 最常见的单词

151. 翻转字符串里的单词.md

给你一个字符串 s ， 逐个翻转字符串中的所有 单词 。

单词 是由非空格字符组成的字符串。s 中使用至少一个空格将字符串中的 单词 分隔开。

请你返回一个翻转 s 中单词顺序并用单个空格相连的字符串。

说明：

```
输入字符串 s 可以在前面、后面或者单词间包含多余的空格。
翻转后单词间应当仅用一个空格分隔。
翻转后的字符串中不应包含额外的空格。
```

示例 1：

输入：s = "the sky is blue"
输出："blue is sky the"

示例 2：

输入：s = " hello world "
输出："world hello"
解释：输入字符串可以在前面或者后面包含多余的空格，但是翻转后的字符不能包括。

示例 3：

输入：s = "a good   example"
输出："example good a"
解释：如果两个单词间有多余的空格，将翻转后单词间的空格减少到只含一个。

示例 4：

输入：s = " Bob   Loves Alice   "
输出："Alice Loves Bob"

示例 5：

输入：s = "Alice does not even like bob"
输出："bob like even not does Alice"

提示：

```
1 <= s.length <= 104
s 包含英文大小写字母、数字和空格 ' '
s 中 至少存在一个 单词
```

进阶：

```
请尝试使用 O(1) 额外空间复杂度的原地解法。
```

```python
class Solution:
    def reverseWords(self, s: str) -> str:
        return ' '.join(s.strip().split()[::-1])
```

Tips

1. string的strip，如果不指定sep，所有white space都会被移除
2. python string是不可变的所以没有原地解法

434. 字符串中的单词数.md

统计字符串中的单词个数，这里的单词指的是连续的不是空格的字符。

请注意，你可以假定字符串里不包括任何不可打印的字符。

示例:

输入: "Hello, my name is John"
输出: 5
解释: 这里的单词是指连续的不是空格的字符，所以 "Hello," 算作 1 个单词。

```
class Solution:
    def countSegments(self, s: str) -> int:
        return len([i for i in s.split(' ') if i.strip()])
```

Tips

直接用字符函数就好，注意"" =0的badcase

500. 键盘行.md

给你一个字符串数组 words ，只返回可以使用在 美式键盘 同一行的字母打印出来的单词。键盘如下图所示。

美式键盘 中：

```
第一行由字符 "qwertyuiop" 组成。
第二行由字符 "asdfghjkl" 组成。
第三行由字符 "zxcvbnm" 组成。
```

American keyboard

示例 1：

输入：words = ["Hello","Alaska","Dad","Peace"]
输出：["Alaska","Dad"]

示例 2：

输入：words = ["omk"]
输出：[]

示例 3：

输入：words = ["adsdf","sfd"]
输出：["adsdf","sfd"]

提示：

```
1 <= words.length <= 20
1 <= words[i].length <= 100
words[i] 由英文字母（小写和大写字母）组成
```

```
class Solution:
    def findWords(self, words: List[str]) -> List[str]:
        res = []
        for i in words:
            j = i.lower()
            if j.strip('qwertyuiop')=='' or j.strip('asdfghjkl')=='' or
j.strip('zxcvbnm')=='':
                res.append(i)
        return res
```

Tips

这个也是长见识了，我一直以为strip只能移除前后的''，这一查才发现strip基本等于replace('abc','')的效果

520. 检测大写字母.md

给定一个单词，你需要判断单词的大写使用是否正确。

我们定义，在以下情况时，单词的大写用法是正确的：

```
全部字母都是大写，比如"USA"。
单词中所有字母都不是大写，比如"leetcode"。
如果单词不只含有一个字母，只有首字母大写， 比如 "Google"。
```

否则，我们定义这个单词没有正确使用大写字母。

示例 1:

输入: "USA"
输出: True

示例 2:

输入: "FlaG"
输出: False

注意: 输入是由大写和小写拉丁字母组成的非空单词。

```
class Solution:
    def detectCapitalUse(self, word: str) -> bool:
        return word.isupper() or word.islower() or (word[0].isupper and
word[1:].islower())
```

Tips

这是一道可以常规按array遍历来解，也可以直接巧用string的function 来进行判断的题目

541. 反转字符串 II.md

给定一个字符串 s 和一个整数 k，从字符串开头算起，每计数至 2k 个字符，就反转这 2k 字符中的前 k 个字符。

```
如果剩余字符少于 k 个，则将剩余字符全部反转。
如果剩余字符小于 2k 但大于或等于 k 个，则反转前 k 个字符，其余字符保持原样。
```

示例 1：

输入：s = "abcdefg", k = 2
输出："bacdfeg"

示例 2：

输入：s = "abcd", k = 2
输出："bacd"

提示：

```
1 <= s.length <= 104
s 仅由小写英文组成
1 <= k <= 104
```

```
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        s = list(s)
        for i in range(0, len(s), 2*k):
            s[i:(i+k)] = reversed(s[i:(i+k)])
        return ''.join(s)
```

Tips

巧用range(start, end, step)

557. 反转字符串中的单词 III.md

给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

示例：

输入："Let's take LeetCode contest"
输出："s'teL ekat edoCteeL tsetnoc"

提示：

> 在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

```
class Solution:
    def reverseWords(self, s: str) -> str:
        return ' '.join([i[::-1] for i in s.split(' ') ])
```

Tips

因为python string是不可变的，所以在原地遍历反转string没有必要，直接用python自带的反转和string split就可以

58. 最后一个单词的长度.md

给你一个字符串 `s`，由若干单词组成，单词前后用一些空格字符隔开。返回字符串中最后一个单词的长度。

单词 是指仅由字母组成、不包含任何空格字符的最大子字符串。

示例 1：

```
输入：s = "Hello World"
输出：5
```

示例 2：

```
输入：s = "   fly me   to   the moon  "
输出：4
```

示例 3：

```
输入：s = "luffy is still joyboy"
输出：6
```

提示：

- `1 <= s.length <= 104`
- `s` 仅有英文字母和空格 `' '` 组成
- `s` 中至少存在一个单词

```python
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        s = s.strip()
        counter=0
        for i in range(len(s)-1, -1, -1):
            if s[i]!=' ':
                counter+=1
            else:
                return counter
        return counter
```

709. 转换成小写字母.md

给你一个字符串 s ，将该字符串中的大写字母转换成相同的小写字母，返回新的字符串。

示例 1：

输入：s = "Hello"
输出："hello"

示例 2：

输入：s = "here"
输出："here"

示例 3：

输入：s = "LOVELY"
输出："lovely"

提示：

```
1 <= s.length <= 100
s 由 ASCII 字符集中的可打印字符组成
```

```python
class Solution:
    def toLowerCase(self, s: str) -> str:
        return s.lower()
```

744. 寻找比目标字母大的最小字母.md

给你一个排序后的字符列表 letters ，列表中只包含小写英文字母。另给出一个目标字母 target，请你寻找在这一有序列表里比目标字母大的最小字母。

在比较时，字母是依序循环出现的。举个例子：

```
如果目标字母 target = 'z' 并且字符列表为 letters = ['a', 'b']，则答案返回 'a'
```

示例：

输入:
letters = ["c", "f", "j"]
target = "a"
输出: "c"

输入:
letters = ["c", "f", "j"]
target = "c"
输出: "f"

输入:
letters = ["c", "f", "j"]
target = "d"
输出: "f"

输入:
letters = ["c", "f", "j"]
target = "g"
输出: "j"

输入:
letters = ["c", "f", "j"]
target = "j"
输出: "c"

输入:
letters = ["c", "f", "j"]
target = "k"
输出: "c"

提示:

```
letters长度范围在[2, 10000]区间内。
letters 仅由小写字母组成，最少包含两个不同的字母。
目标字母target 是一个小写字母。
```

```python
class Solution:
    def nextGreatestLetter(self, letters: List[str], target: str) -> str:
        for i in letters:
            if i > target:
                return i
        return letters[0]
```

Tips

python字符可以直接进行大小比较，具体大小是按照ord来进行比较的，小写在前，大写在后

技巧是letters已经是排序的了，所以如果找不到比target更大的字母直接返回第一个就好

748. 最短补全词.md

给你一个字符串 licensePlate 和一个字符串数组 words ，请你找出并返回 words 中的 最短补全词 。

补全词 是一个包含 licensePlate 中所有的字母的单词。在所有补全词中，最短的那个就是 最短补全词 。

在匹配 licensePlate 中的字母时:

```
忽略 licensePlate 中的 数字和空格 。
不区分大小写。
如果某个字母在 licensePlate 中出现不止一次，那么该字母在补全词中的出现次数应当一致或者更多。
```

例如：licensePlate = "aBc 12c"，那么它的补全词应当包含字母 'a'、'b' （忽略大写）和两个 'c' 。可能的 补全词 有 "abccdef"、"caaacab" 以及 "cbca" 。

请你找出并返回 words 中的 最短补全词 。题目数据保证一定存在一个最短补全词。当有多个单词都符合最短补全词的匹配条件时取 words 中 最靠前的 那个。

示例 1:

输入：licensePlate = "1s3 PSt", words = ["step", "steps", "stripe", "stepple"]
输出："steps"
解释：最短补全词应该包括 "s"、"p"、"s"（忽略大小写） 以及 "t"。
"step" 包含 "t"、"p"，但只包含一个 "s"，所以它不符合条件。
"steps" 包含 "t"、"p" 和两个 "s"。
"stripe" 缺一个 "s"。
"stepple" 缺一个 "s"。
因此，"steps" 是唯一一个包含所有字母的单词，也是本例的答案。

示例 2：

输入：licensePlate = "1s3 456", words = ["looks", "pest", "stew", "show"]
输出："pest"
解释：licensePlate 只包含字母 "s" 。所有的单词都包含字母 "s" ，其中 "pest"、"stew"、和 "show" 三者最短。
答案是 "pest" ，因为它是三个单词中在 words 里最靠前的那个。

示例 3：

输入：licensePlate = "Ah71752", words =
["suggest","letter","of","husband","easy","education","drug","prevent","writer","old"]
输出："husband"

示例 4：

输入：licensePlate = "OgEu755", words =
["enough","these","play","wide","wonder","box","arrive","money","tax","thus"]
输出："enough"

示例 5：

输入：licensePlate = "iMSlpe4", words =
["claim","consumer","student","camera","public","never","wonder","simple","thought","use"]
输出："simple"

提示：

```
1 <= licensePlate.length <= 7
licensePlate 由数字、大小写字母或空格 ' ' 组成
1 <= words.length <= 1000
1 <= words[i].length <= 15
words[i] 由小写英文字母组成
```

```python
class Solution:
    def shortestCompletingWord(self, licensePlate: str, words: List[str]) -> str:
        dic = defaultdict(int)
        for i in licensePlate:
            if i.strip() and not i.isdigit():
```

```
                dic[i.lower()]+=1

        def check(word, dic):
            for i in word:
                if i.lower() in dic:
                    dic[i.lower()]-=1

            return all((i<=0 for i in dic.values()))

        ans = None
        for word in words:
            if (ans is  None or len(word)<len(ans)) and check(word, dic.copy()):
                ans =word
        return ans
```

Tips

1. 需要注意只有小写字母被统计
2. 且最短补全词是只要包含这些字母就好，出现次数可以大于
3. 注意dict不要修改蓄意要穿copy
4.

796. 旋转字符串.md

给定两个字符串, A 和 B。

A 的旋转操作就是将 A 最左边的字符移动到最右边。 例如, 若 A = 'abcde'，在移动一次之后结果就是'bcdea' 。如果在若干次旋转操作之后，A 能变成B，那么返回True。

示例 1:
输入: A = 'abcde', B = 'cdeab'
输出: true

示例 2:
输入: A = 'abcde', B = 'abced'
输出: false

注意:

```
A 和 B 长度不超过 100。
```

```
class Solution:
    def rotateString(self, s: str, goal: str) -> bool:
        if len(s)!=len(goal):
            return False
        return goal in s+s
```

Tips

总觉得还看过这个技巧但是想不起来了。其实就是旋转操作的所有解被包含在S+S里面，所以只要看判断target是否是S+S的子序列即可

8. 字符串转换整数 (atoi).md

请你来实现一个 myAtoi(string s) 函数，使其能将字符串转换成一个 32 位有符号整数（类似 C/C++ 中的 atoi 函数）。

函数 myAtoi(string s) 的算法如下：

```
读入字符串并丢弃无用的前导空格
检查下一个字符（假设还未到字符末尾）为正还是负号，读取该字符（如果有）。 确定最终结果是负数还是正数。 如果两者都不存在，则假定结果为正。
读入下一个字符，直到到达下一个非数字字符或到达输入的结尾。字符串的其余部分将被忽略。
将前面步骤读入的这些数字转换为整数（即，"123" -> 123， "0032" -> 32）。如果没有读入数字，则整数为 0 。必要时更改符号（从步骤 2 开始）。
如果整数数超过 32 位有符号整数范围 [−231， 231 − 1] ，需要截断这个整数，使其保持在这个范围内。具体来说，小于 −231 的整数应该被固定为 −231 ，大于 231 − 1 的整数应该被固定为 231 − 1 。
返回整数作为最终结果。
```

注意：

```
本题中的空白字符只包括空格字符 ' ' 。
除前导空格或数字后的其余字符串外，请勿忽略 任何其他字符。
```

示例 1：

输入：s = "42"
输出：42
解释：加粗的字符串为已经读入的字符，插入符号是当前读取的字符。
第 1 步："42"（当前没有读入字符，因为没有前导空格）
^
第 2 步："42"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
^
第 3 步："42"（读入 "42"）
^
解析得到整数 42 。
由于 "42" 在范围 [-231, 231 - 1] 内，最终结果为 42 。

示例 2：

输入：s = "   -42"
输出：-42
解释：
第 1 步："   -42"（读入前导空格，但忽视掉）
         ^
第 2 步："   -42"（读入 '-' 字符，所以结果应该是负数）
          ^
第 3 步："   -42"（读入 "42"）
           ^
解析得到整数 -42 。
由于 "-42" 在范围 [-231, 231 - 1] 内，最终结果为 -42 。

示例 3：

输入：s = "4193 with words"
输出：4193
解释：
第 1 步："4193 with words"（当前没有读入字符，因为没有前导空格）
       ^
第 2 步："4193 with words"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
       ^
第 3 步："4193 with words"（读入 "4193"；由于下一个字符不是一个数字，所以读入停止）
           ^
解析得到整数 4193 。
由于 "4193" 在范围 [-231, 231 - 1] 内，最终结果为 4193 。

示例 4：

输入：s = "words and 987"
输出：0
解释：
第 1 步："words and 987"（当前没有读入字符，因为没有前导空格）
       ^
第 2 步："words and 987"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
       ^
第 3 步："words and 987"（由于当前字符 'w' 不是一个数字，所以读入停止）
       ^
解析得到整数 0 ，因为没有读入任何数字。
由于 0 在范围 [-231, 231 - 1] 内，最终结果为 0 。

示例 5：

输入：s = "-91283472332"
输出：-2147483648
解释：
第 1 步："-91283472332"（当前没有读入字符，因为没有前导空格）
       ^
第 2 步："-91283472332"（读入 '-' 字符，所以结果应该是负数）
        ^

第 3 步："-91283472332"（读入 "91283472332"）
           ^
解析得到整数 -91283472332 。
由于 -91283472332 小于范围 [-231, 231 - 1] 的下界，最终结果被截断为 -231 = -2147483648 。

提示：

```
0 <= s.length <= 200
s 由英文字母（大写和小写）、数字（0-9）、' '、'+'、'-' 和 '.' 组成
```

```python
class Solution:
    def myAtoi(self, s: str) -> int:
        INT_MAX = 0x7fffffff
        INT_MIN = -1-0x7fffffff
        if not s:
            return 0

        s = s.strip()

        if not s:
            return 0

        flag = 1
        if s[0] in ['+','-']:
            if s[0]=='-':
                flag = -1
            s = s[1:]

        if not s or not(s[0].isdigit()):
            return 0

        result = []
        for i,num in enumerate(s):
            if num.isdigit():
                result.append(num)
            else:
                break

        num = 0
        for i in result:
            num*=10
            num+=int(i)
        num*=flag

        if num>INT_MAX:
```

```
            return INT_MAX
        elif num<INT_MIN:
            return INT_MIN
        else:
            return num
```

804. 唯一摩尔斯密码词.md

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。
```

```
class Solution:
    def uniqueMorseRepresentations(self, words: List[str]) -> int:
        code=[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-
",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--
","--.."]

        def translate(word):
            ans =[]
            for i in word:
                ans.append(code[ord(i)-97])
            return ''.join(ans)
        ans = set()
        for word in words:
            ans.add(translate(word))
        return len(ans)
```

Tips

需要用到ord('a')最小是97

806. 写字符串需要的行数.md

我们要把给定的字符串 S 从左到右写到每一行上，每一行的最大宽度为100个单位，如果我们在写某个字母的时候会使这行超过了100 个单位，那么我们应该把这个字母写到下一行。我们给定了一个数组 widths ，这个数组 widths[0] 代表 'a' 需要的单位， widths[1] 代表 'b' 需要的单位，..., widths[25] 代表 'z' 需要的单位。

现在回答两个问题：至少多少行能放下S，以及最后一行使用的宽度是多少个单位？将你的答案作为长度为2的整数列表返回。

示例 1:
输入:
widths = [10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]
S = "abcdefghijklmnopqrstuvwxyz"
输出: [3, 60]
解释:
所有的字符拥有相同的占用单位10。所以书写所有的26个字母，
我们需要2个整行和占用60个单位的一行。

示例 2:
输入:
widths = [4,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]
S = "bbbcccdddaaa"
输出: [2, 4]
解释:
除去字母'a'所有的字符都是相同的单位10，并且字符串 "bbbcccdddaa" 将会覆盖 9 * 10 + 2 * 4 = 98 个单位.
最后一个字母 'a' 将会被写到第二行，因为第一行只剩下2个单位了。
所以，这个答案是2行，第二行有4个单位宽度。

注:

```
字符串 S 的长度在 [1, 1000] 的范围。
S 只包含小写字母。
widths 是长度为 26的数组。
widths[i] 值的范围在 [2, 10]。
```

```python
class Solution:
    def numberOfLines(self, widths: List[int], s: str) -> List[int]:
        row = 1
        cursum = 0
        for i in s:
            width = widths[ord(i)-97]
            if cursum + width > 100:
                cursum = width
                row +=1
            else:
                cursum+=width
        return [row, cursum]
```

Tips

这题好好身审题就成了。。。

819. 最常见的单词.md

给定一个段落 (paragraph) 和一个禁用单词列表 (banned)。返回出现次数最多，同时不在禁用列表中的单词。

题目保证至少有一个词不在禁用列表中，而且答案唯一。

禁用列表中的单词用小写字母表示，不含标点符号。段落中的单词不区分大小写。答案都是小写字母。

示例:

输入:
paragraph = "Bob hit a ball, the hit BALL flew far after it was hit."
banned = ["hit"]
输出: "ball"
解释:
"hit" 出现了3次，但它是一个禁用的单词。
"ball" 出现了2次 (同时没有其他单词出现2次)，所以它是段落里出现次数最多的，且不在禁用列表中的单词。
注意，所有这些单词在段落里不区分大小写，标点符号需要忽略（即使是紧挨着单词也忽略， 比如 "ball,"），
"hit"不是最终的答案，虽然它出现次数更多，但它在禁用单词列表中。

提示：

```
1 <= 段落长度 <= 1000
0 <= 禁用单词个数 <= 100
1 <= 禁用单词长度 <= 10
答案是唯一的，且都是小写字母（即使在 paragraph 里是大写的，即使是一些特定的名词，答案都是小写的。）
paragraph 只包含字母、空格和下列标点符号!?',;.
不存在没有连字符或者带有连字符的单词。
单词里只包含字母，不会出现省略号或者其他标点符号。
```

```python
class Solution:
    def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
        from collections import Counter
        banned = set(banned)
        for c in "!?',;.":
            paragraph=paragraph.replace(c,' ')
        dic = Counter(filter(lambda x:len(x)>0, paragraph.lower().split(' ')))
        for key,_ in sorted(dic.items(), key=lambda x: x[1], reverse=True):
            if key not in banned:
                return key
```

Tips

这题全在细节里

1. 要剔除标点符号，切剔除后要补‘’
2. 直接用Counter不行，因为上述split之后可能会出现空字符，要过滤
3. 注意只保留小写

824. 山羊拉丁文.md

给定一个由空格分割单词的句子 S。每个单词只包含大写或小写字母。

我们要将句子转换为 “Goat Latin”（一种类似于 猪拉丁文 - Pig Latin 的虚构语言）。

山羊拉丁文的规则如下：

如果单词以元音开头（a, e, i, o, u），在单词后添加"ma"。
例如，单词"apple"变为"applema"。

如果单词以辅音字母开头（即非元音字母），移除第一个字符并将它放到末尾，之后再添加"ma"。
例如，单词"goat"变为"oatgma"。

根据单词在句子中的索引，在单词最后添加与索引相同数量的字母'a'，索引从1开始。
例如，在第一个单词后添加"a"，在第二个单词后添加"aa"，以此类推。

返回将 S 转换为山羊拉丁文后的句子。

示例 1:

输入: "I speak Goat Latin"
输出: "Imaa peaksmaaa oatGmaaaa atinLmaaaaa"

示例 2:

输入: "The quick brown fox jumped over the lazy dog"
输出: "heTmaa uickqmaaa rownbmaaaa oxfmaaaaa umpedjmaaaaaa overmaaaaaaa hetmaaaaaaaa azylmaaaaaaaaa ogdmaaaaaaaaaa"

说明:

S 中仅包含大小写字母和空格。单词间有且仅有一个空格。
1 <= S.length <= 150。

```python
class Solution:
    def toGoatLatin(self, sentence: str) -> str:
        ans = []
        for p, i in enumerate(sentence.split(' ')):
            if i[0] in 'aeiouAEIOU':
                i+='ma'
            else:
                i = i[1:] + i[0] + 'ma'
            i+= 'a'*(p+1)
            ans.append(i)
        return ' '.join(ans)
```