数字图像隐写术 F3，F4，F5 的分析与研究

信息科学与技术学院　信息安全第三小组

摘 要

数字图像隐写术是将文本信息隐藏在数字图像中的技术。本文对数字图像隐写术中的典型算法 F3，F4，F5 隐写术进行了算法分析和实验仿真。在此基础上，对三者的算法分析结果进行优缺点和效率对比，对仿真结果进行统计和隐写系统分析。研究发现，在 F3，F4 基础上改进而来的 F5 算法引入混洗技术密钥提高了安全性，引入矩阵编码技术提高隐藏信息嵌入效率。F5 嵌入信息后所得的图像，质量均匀，提高了对算法安全分析的统计对抗能力。

关键词　隐写术；F3；F4；F5

Abstract

Digital image steganography is a kind of technology to hide text information in digital image. In this paper, aimed at typical digital image steganographic algorithms F3, F4, F5 algorithm analysis and experimental simulation are carried out. On this basis, the analysis results of these three algorithms are compared in terms of advantages, disadvantages and efficiency. And statistical analysis and steganographic system analysis on the simulation results are done as well. It's found that F5 steganographic algorithm, improved based on F3 and F4, introduces shuffling technology and key to improve the security, and matrix coding technology to improve the efficiency of information hiding. When it comes to embedded image obtained by F5 steganographic algorithm, its feature of uniform quality improves the combat capability of the algorithm statistical security analysis.

Key words　steganography；F3；F4；F5

1　引言

隐写术是一门关于信息隐藏的技巧与科学，所谓信息隐藏指的是不让除预期的接收者之外的任何人知晓信息的传递事件或者信息的内容。由于载体文件

相对隐秘文件的大小（指数据含量，以比特计）越大，隐藏后者就越加容易，数字图像（包含有大量的数据）在因特网和其他传媒上被广泛用于隐藏消息。F3，F4，F5 隐写术便是数字图像隐写术中的典型代表，后者皆是依托前者，经过优化改进得到，且具有广泛的应用价值。算法的核心思想是，通过对 JPEG 图像的 DCT 系数的最低比特位的修改，以达到秘密嵌入指定文本信息的作用。

2 算法分析

2.1 F3 隐写术

2.1.1 F3 实现步骤

（1）偶数代表消息 0，奇数代表 1；

（2）在每个非 0 的 DCT 系数中嵌入 1 比特秘密信息。如 DCT 系数的 LSB 与秘密信息不同，则将 DCT 系数的绝对值减 1，符号不变；

（3）如果嵌入过程产生值为 0 的 DCT 系数，则嵌入无效，需在下一个 DCT 系数上重新嵌入；

（4）提取信息时，取出不为 0 的 DCT 系数的 LSB 即可。

2.1.2 F3 原理图

使用 F3 隐写术嵌入数据时，LSB 的转移和代表信息含义如图 1 所示：

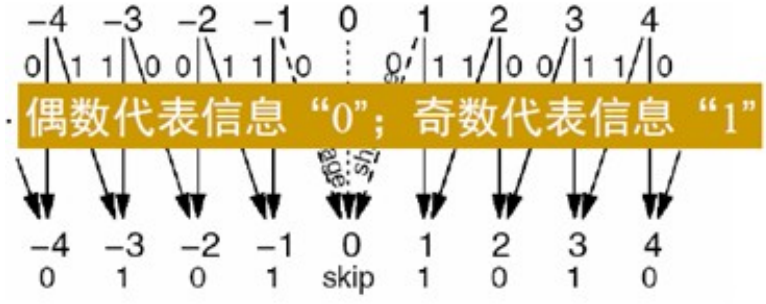

图 1: F3 原理图

2.1.3 F3 伪代码描述

$$
\begin{aligned}
&Function\ of\ F3\ Steganography\\
&for\ i \leftarrow 1\ to\ m\\
&for\ j \leftarrow 1\ to\ n\\
&if\ DCT_j\ is\ odd\ then\ DCT_j \leftarrow 1\ else\ DCT_j \leftarrow 0\\
&if\ DCT_j \neq DATA_i\ then\ |DCT_j| \leftarrow |DCT_j| - 1\\
&if\ DCT_j \neq 0\ then\ break
\end{aligned}
$$

2.2 F4 隐写术

2.2.1 F4 实现步骤

（1）正偶数和负奇数代表消息 0，负偶数和正奇数代表 1；

（2）在每个非 0 的 DCT 系数中嵌入 1 比特秘密信息。如 DCT 系数的 LSB 与秘密信息不同，则将 DCT 系数的绝对值减 1，符号不变；

（3）如果嵌入过程产生值为 0 的 DCT 系数，则嵌入无效，需在下一个 DCT 系数上重新嵌入；

（4）提取信息时，取出不为 0 的 DCT 系数的 LSB 即可。

2.2.2 F4 原理图

使用 F4 隐写术嵌入数据时，LSB 的转移和代表信息含义如图 2 所示：

图 2: F4 原理图

2.2.3 F4 伪代码描述

Function of F4 Steganography

$for\ i \leftarrow 1\ to\ m$

$for\ j \leftarrow 1\ to\ n$

$if\ DCT_j > 0\ then$

$\quad if\ DCT_j\ is\ odd\ then\ DCT_j \leftarrow 1\ else\ DCT_j \leftarrow 0$

$end\ if$

$if\ DCT_j < 0\ then$

$\quad if\ DCT_j\ is\ even\ then\ DCT_j \leftarrow 1\ else\ DCT_j \leftarrow 0$

$end\ if$

$if\ DCT_j \neq DATA_i\ then\ |DCT_j| \leftarrow |DCT_j| - 1$

$if\ DCT_j \neq 0\ then\ break$

2.3 F5 隐写术

2.3.1 F5 实现步骤

（1）正偶数和负奇数代表消息 0，负偶数和正奇数代表 1；

（2）以密码为随机化种子，混洗 DCT 系数的下标数组；

（3）根据欲嵌入的秘密信息长度计算得到嵌入信息所使用的三元组 $(1, n, k)$;

（4）采用矩阵编码方式，根据下标数组，每次提取 n 个非 0 的 DCT 系数，由异或关系嵌入 k 比特秘密信息；

（5）如果嵌入过程产生值为 0 的 DCT 系数，则嵌入无效，需在重新嵌入。

（6）提取信息时，根据混洗下标数组，每次取出 n 个不为 0 的 DCT 系数，将其加权异或即可得到 k 比特信息。

2.3.2 F5 解释性语言描述

F5 嵌入秘密信息的解释性语言描述见附录 1，提取信息部分关键代码见附录 2。

3 仿真实现

通过计算机编程仿真实现隐写术，得到下面 4 种情况下三种隐写术的效果图。其中，所需嵌入的英文数据为 2.49KB，中文数据为 3.09KB，灰度图像及

彩色图像大小都为 512×512 像素。每组仿真结果展示的四幅图像，从左至右依次为原图、F3 隐写图、F4 隐写图、F5 隐写图。

隐藏英文数据，灰度图像作载体仿真图如图 3 所示：

图 3: 仿真效果图 1

隐藏英文数据，彩色图像作载体仿真图如图 4 所示：

图 4: 仿真效果图 2

隐藏中文数据，灰度图像作载体仿真图如图 5 所示：

图 5: 仿真效果图 3

隐藏中文数据，彩色图像作载体仿真图如图 6 所示：

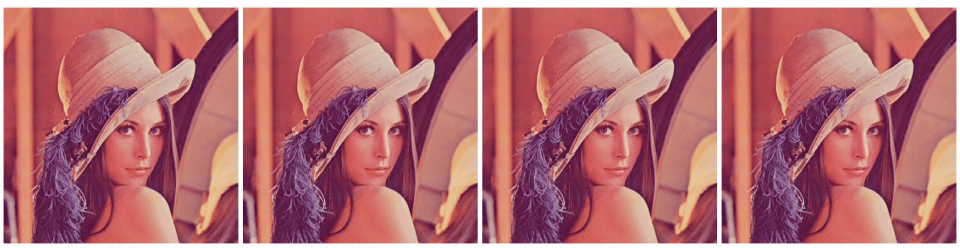

图 6: 仿真效果图 4

4 对比研究

4.1 优缺点概括

4.1.1 F3 概括

优点 保持了 DCT 系数统计分布关于 0 的对称性；在 DCT 系数 1 或者 -1 中嵌入消息 0 得到 0，此时检测器不能分辨 DCT 系数中消息嵌入产生的 0 和未使用的 0，从而不能正确提取消息位，因而必须重传这一消息 0。

缺点 嵌入了更多的零，这一异常可以被利用于隐写分析，重传导致 DCT 系数中偶数明显增多 (除 0 外)。原始图像一般来说 DCT 系数中奇数总数多于偶数总数 (不包括零)，这样很容易区分开原始图像和载密图像。

4.1.2 F4 概括

优点 重传的消息比特 0 和 1 的个数近似相等，很难被利用与隐写分析。容易区分开原始图像和载密图像。使直方图特性得到了保持。

缺点 隐写是顺序嵌入的，这就使得 LSB 的修改集中在图像的某一部分，可能导致图像质量的不均匀。当嵌入信息时，对系数的 LSB 更改的概率为 1/2，所以在嵌入较多信息时，系数的更改会比较多，检测者据此很可能发现秘密信息的存在。

4.1.3 F5 概括

优点 采用矩阵编码来嵌入信息，在 LSB 改变很小的情况下嵌入大量的隐藏信息，嵌入效率高，有极强的鲁棒性，同时矩阵编码可以使得 RS 统计分析不能正确测出嵌入信息的长度。

缺点 对 DCT 直方图的改变方式是固定的，并且在潜入过程中会使 0 的 DCT 系数增加。

4.2 效率对比分析

理论分析 F3，F4，F5 隐写术的复杂度及效率，假设所需嵌入信息为 m 比特，DCT 所能提供的 LSB 为 n 比特，则结果如表 1 所示：

表 1: 效率对比分析表

隐写术	时间复杂度	空间复杂度	嵌入率	嵌入效率
F3	$O(m)$	$O(m+n)$	1	1
F4	$O(m)$	$O(m+n)$	1	1
F5	$O(m+n)$	$O(m+n)$	k/n	$(n+1)*k/n$

对于 F5 隐写术，不同的矩阵编码方式及不同的三元组 $(1, n, k)$ 的选择，使其变化密度、嵌入率和嵌入效率之间的关系如表 2 所示：

表 2: 矩阵编码分析表

k	n	改变比特率 (%)	嵌入率 (%)	嵌入效率
1	1	50.00	100.00	2
2	3	25.00	66.67	2.67
3	7	12.50	42.86	3.43
4	15	6.25	26.67	4.27
5	31	3.12	16.13	5.16
6	63	1.56	9.52	6.09
7	127	0.78	5.51	7.06
8	255	0.39	3.14	8.03
9	511	0.20	1.76	9.02

4.3 安全性分析

4.3.1 DCT 系数统计

针对本文第 3 节仿真实验中的 4 种情况，统计其值为 -8 $+8$ 的 DCT 系数，得到统计直方图。同样，每组仿真结果展示的四幅图像，从左至右依次为原图直方图、F3 直方图、F4 直方图、F5 直方图。

隐藏英文数据，灰度图像作载体，生成的载密图像的 DCT 系数直方图如图 7 所示：

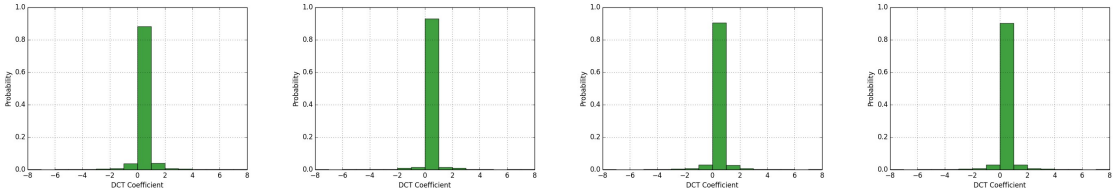

图 7: DCT 系数直方图 1

隐藏英文数据，彩色图像作载体，生成的载密图像的 DCT 系数直方图如图 8 所示：

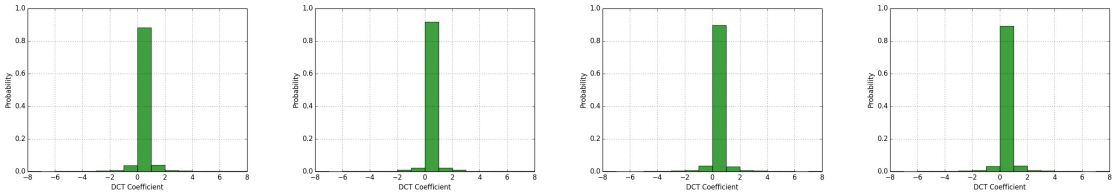

图 8: DCT 系数直方图 2

隐藏中文数据，灰度图像作载体，生成的载密图像的 DCT 系数直方图如图 9 所示：

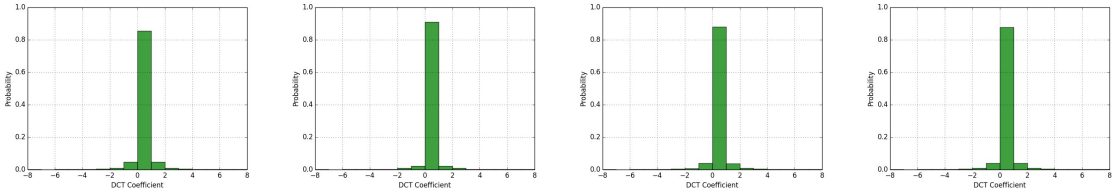

图 9: DCT 系数直方图 3

隐藏中文数据，彩色图像作载体，生成的载密图像的 DCT 系数直方图如图 10 所示：

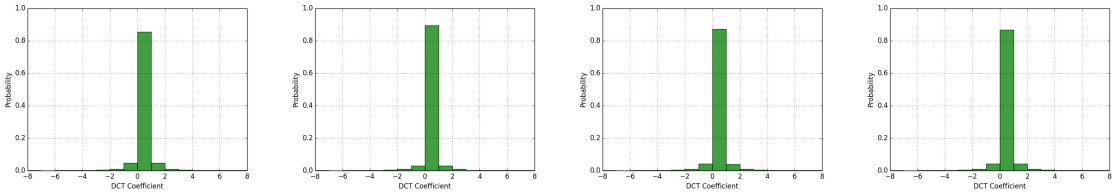

图 10: DCT 系数直方图 4

4.3.2 隐写系统分析

针对 F5 的攻击技术除了 J.Fridrich 提出的专用隐写分析算法之外，大都使用通用隐写分析算法，但是对于低嵌入率载密图像的检测效果也都不是很理想。专用隐写分析算法由于特征的单一性因此没能很好的解决虚警率和漏检率之间的矛盾，而通用隐写分析算法虽然有很强的适应性, 但是算法复杂度很大，而且算法的设计针对性不是很强。

在对于唯载密攻击，无法知道载体图像的统计特性，针对 F5 的隐写分析算法，J.Fridrich 提出了一种基于裁减技术估计载体图像，并根据最小平方误差准

则进行嵌入率估计的方法。J.Fridrich 等人认为对载体/载密图像裁减上 4 行左 4 列，平滑滤波后再利用相同的量化表压缩能较好地近似载体图像。

4.3.3 F5 密钥分析

F5 中利用密钥产生了随机数序列，作用有两个。一是使用产生乱序的下标数组，混洗 DCT 系数；二是初始化需要隐藏的信息，对其每字节，使用一个随机数与其异或，在矩阵编码时使用初始化后的数据嵌入图像。

由于产生随机数序列的伪随机函数是单元函数，如果第三方想要通过字典攻击暴力破解图像提取隐藏信息，则其需要枚举的密钥空间为 INT 型，即 [-2147483648,2147483647]。如果采用多元函数产生随机化序列，即随机化序列的每一个数由前面多个数共同计算得到，可以认为强行提取图像所含信息，对时间空间开销而言是不可实现的。

5 结论

本文对 JPEG 图像隐写术中典型的 F3，F4，F5 隐写与检测算法进行了算法研究、实验仿真和对比分析。在算法的研究方面：理解并分析了 F3，F4，F5 的算法思想及实现方法，对比研究了 F3，F4，F5 算法之间的差异和优缺点。在实验仿真方面：对不同语言的文本信息，使用 F3，F4，F5 算法，分别嵌入到不同分辨率及类型的图像中，进行分析和比较。

可以得出以下结论：F3、F4、F5 隐写术都具有将信息隐藏在图片中的作用，并且性能较优。F3，F4 直接对 DCT 系数的 LSB 进行修改以嵌入信息，F5 先通过密码作为随机化种子对 DCT 系数进行混洗，再利用矩阵编码对 DCT 系数的 LSB 嵌入信息。嵌入后依次按照提取规则，对每种隐写术产生的隐写图像都进行了嵌入信息的还原。F5 相对前两者的主要区别在于抗检测：混洗技术使卡方检测不易进行；密码引入加强了数据的安全性；嵌入相同信息，需要改变的位少，但整体能够嵌入的信息长度不够。所以，F5 隐写术能在不损坏图像的前提下，较好的隐藏信息。对于加强安全传输，防第三方窥视具有较大的优势，应用前景广阔。

6 参考文献

[1] 于丽芳. JPEG 图像隐写术研究 [D]. 北京交通大学,2013.

[2] 陈志立. 语言隐写术的分析与设计研究 [D]. 中国科学技术大学,2009.

[3] 宋晓麟, 李才明, 张锐. 信息隐藏的重要分支——数字水印和隐写术 [J]. 内蒙古石油化工,2006, 11 : 33 – 35.

[4] 葛秀慧, 胡爱华, 田浩, 王嘉祯. 隐写术的研究与应用 [J]. 计算机应用与软件,2007, 11 : 57 − 60.

[5] 邵凯. 基于图像的信息隐藏与隐写术技术研究 [D]. 西安电子科技大学,2008.

[6] 郎荣玲, 夏煜, 郅艳, 戴冠中. 几类典型隐写术分析算法的分析与评价 [J]. 中国图象图形学报,2004, 02 : 124 − 131.

[7] 王超. 大嵌入率矩阵编码及其在隐写术中的应用 [D]. 解放军信息工程大学,2012.

[8] 王少宾, 张定会. JPEG 图像的信息隐写术与隐写分析 [J]. 计算机应用与软件,2010, 11 : 250 − 254 + 275.

[9] 张卫明, 王超, 程森. 隐写术中的编码模型及方法研究 [J]. 信息网络安全,2011, 11 : 8 − 13.

[10] 王岩岩, 武亚菲. 隐写术的应用及安全性研究 [J]. 计算机时代,2012, 03 : 4 − 6.

[11] 傅瑜. 数字图像隐写算法安全性与性能优化研究 [D]. 北京邮电大学,2010.

[12] 韩涛. 隐写编码及其在图像隐写术中的应用 [D]. 解放军信息工程大学,2011.

[13] 崔忠立, 王嘉祯. 一种替换类隐写术算法的统一模型 [J]. 计算机工程与应用,2006, 13 : 127 − 129.

[14] 李浩光. 几种基于 JPEG 图像的隐写术比较研究 [J]. 电脑编程技巧与维护,2013, 08 : 108 − 110.

附录

附录 1：F5 嵌入信息关键代码

```
def embedData5(self):
    logger.info('embedding data in image by f5')
    self.permutate()
    self.matrixCode()

def permutate(self):
    logger.info('permutating')
    random.seed(self.password)
    size=len(self.iJpegE.coeff)
    self.shuffle=[i for i in range(size)]
    for i in range(size):
        index=self.getInt(size)
        size-=1
        self.shuffle[size],self.shuffle[index]=self.
            shuffle[index],self.shuffle[size]

def matrixCode(self):
    logger.info('matrix coding')
    coeff=self.iJpegE.coeff
    coeff_num=len(coeff)
    one,zero=0,0
    for i,j in enumerate(coeff):
        if i%64==0: continue
        elif j==1 or j==-1: one+=1
        elif j==0: zero+=1
    other=coeff_num-one-zero-coeff_num//64
    expect=int(0.5*one)+other
    byte_embed=len(self.data)
    if byte_embed>0x7fffff: byte_embed=0x7fffff
    for i in range(1,10):
        n=(1<<i)-1
        use=(expect//n)*i/8
        if use<byte_embed+4:
```

```
            i=i-1
            break
    k=i
    n=(1<<k)-1
    if n==0:
        logger.info('file not suitable')
        n=1
    else: logger.info('using (1,%d,%d) code'%(n,k))
    logger.info('totally %d bytes to embed by f5'%
       byte_embed)
    byte_embed|=k<<24
    for i in range(4): byte_embed^=self.getByte()<<i*8
    bit_embed=byte_embed&1
    byte_embed>>=1
    need_embed=31
    data_index=0
    shuffle_pos=0
    for j in self.shuffle:
        shuffle_pos+=1
        if j%64==0 or coeff[j]==0: continue
        if coeff[j]>0 and (coeff[j]&1)!=bit_embed: coeff[
           j]-=1
        elif coeff[j]<0 and (coeff[j]&1)==bit_embed:
           coeff[j]+=1
        if coeff[j]!=0:
            if need_embed==0: break;
            bit_embed=byte_embed&1
            byte_embed>>=1
            need_embed-=1
    try:
        while True:
            kbits_embed=0
            for i in range(k):
                if need_embed==0:
                    if data_index>=len(self.data): break
                    byte_embed=ord(self.data[data_index])
                       ^self.getByte()
```

```
                    data_index+=1
                    need_embed=8
                bit_embed=byte_embed&1
                byte_embed>>=1
                need_embed-=1
                kbits_embed|=bit_embed<<i
            refer=create(0,n)
            for i in range(n):
                while self.shuffle[shuffle_pos]%64==0 or
                    coeff[self.shuffle[shuffle_pos]]==0:
                    shuffle_pos+=1
                refer[i]=self.shuffle[shuffle_pos]
                shuffle_pos+=1
            while True:
                fhash=0
                for i,j in enumerate(refer):
                    if coeff[j]>0: tmp=coeff[j]&1
                    else: tmp=coeff[j]&1^1
                    if tmp==1: fhash^=i+1
                d=fhash^kbits_embed
                if d==0: break
                s=refer[d-1]
                if coeff[s]>0: coeff[s]-=1
                elif coeff[s]<0: coeff[s]+=1
                if coeff[s]==0:
                    refer[d-1:d]=[]
                    while self.shuffle[shuffle_pos]%64==0
                        or coeff[self.shuffle[shuffle_pos
                        ]]==0:
                        shuffle_pos+=1
                    refer.append(self.shuffle[shuffle_pos
                        ])
                    shuffle_pos+=1
                else: break
            if data_index>=len(self.data) and need_embed
                ==0: break
    except IndexError:
```

```
        logger.info('data is too long')
```

附录 2：F5 提取信息关键代码

```
def extractData5(self):
    logger.info('extracting data in image by f5')
    self.permutate()
    self.matrixDecode()

def permutate(self):
    logger.info('permutating')
    random.seed(self.password)
    size=len(self.iJpegD.coeff)
    self.shuffle=[i for i in range(size)]
    for i in range(size):
        index=self.getInt(size)
        size-=1
        self.shuffle[size],self.shuffle[index]=self.
            shuffle[index],self.shuffle[size]

def matrixDecode(self):
    logger.info('matrix decoding')
    coeff=self.iJpegD.coeff
    i,shuffle_pos=0,-1
    finish,length=0,0
    need_extract=0
    byte_extract=0
    while i<32:
        shuffle_pos+=1
        shuffle_index=self.shuffle[shuffle_pos]
        if shuffle_index%64==0: continue
        c=coeff[shuffle_index-shuffle_index%64+ZAGZIG[
            shuffle_index%64]]
        if c==0: continue
        if c>0 and c&1: length|=1<<i
        if c<0 and c&1^1: length|=1<<i
        i+=1
    for i in range(4): length^=self.getByte()<<i*8
```

```
k=length>>24&0xf
length=length&0x7fffff
n=(1<<k)-1
logger.info('using (1,%d,%d) code'%(n,k))
logger.info('totally %d bytes to extract by f5'%
    length)
while finish<length:
    i=0
    refer=create(0,n)
    fhash=0
    while i<n:
        shuffle_pos+=1
        shuffle_index=self.shuffle[shuffle_pos]
        if shuffle_index%64==0: continue
        c=coeff[shuffle_index-shuffle_index%64+ZAGZIG
            [shuffle_index%64]]
        if c==0: continue
        if c>0 and c&1: fhash^=i+1
        if c<0 and c&1^1: fhash^=i+1
        i+=1
    for i in range(k):
        byte_extract|=(fhash>>i&1)<<need_extract
        need_extract+=1
        if need_extract==8:
            aaaa=byte_extract
            byte_extract^=self.getByte()
            self.out.write(chr(byte_extract&0xff))
            need_extract=0
            byte_extract=0
            finish+=1
```