2021国赛CISCN 初赛 部分RE writeup

原创

Mer10t 于 2021-09-14 14:57:17 发布 27 收藏

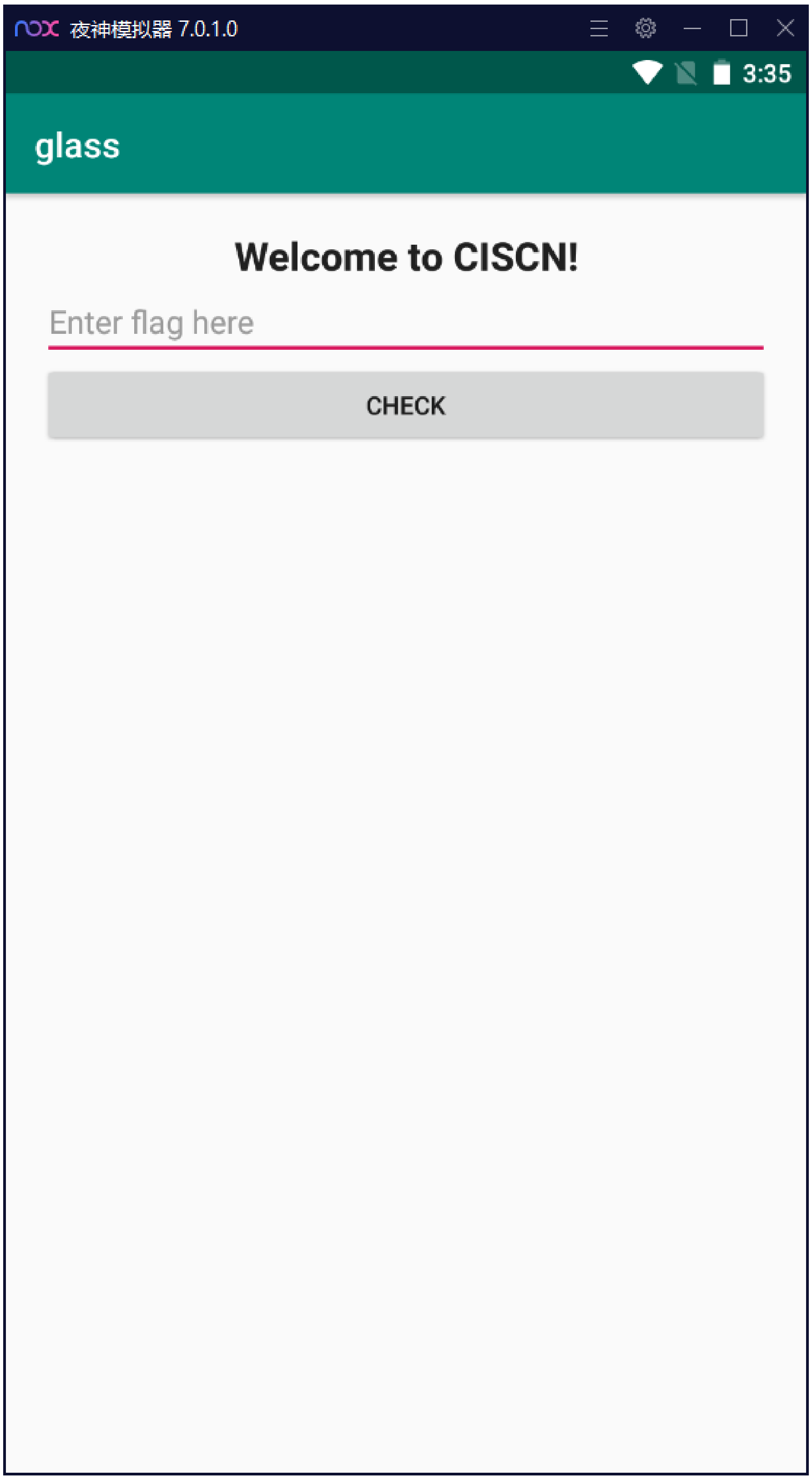
文章标签： 安全

glass

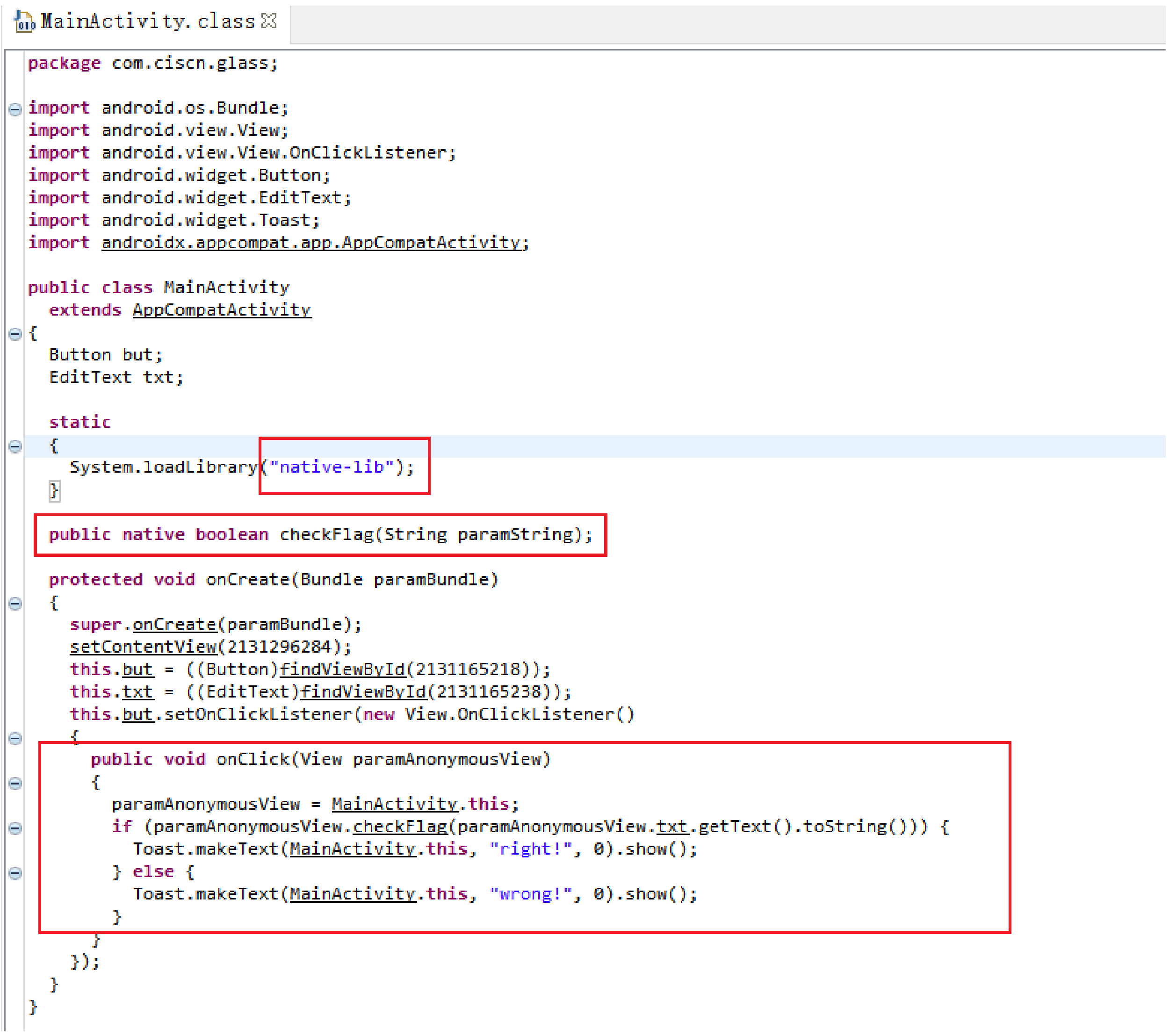
```
题目名称：glass
题目描述：Reverse sign in，flag形式为"CISCN{XXXXX}"
```

首先查看apk

界面如下



用AndroidKiller打开，反编译成java源码



MainActivity.class

```java
package com.ciscn.glass;

import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity
  extends AppCompatActivity
{
  Button but;
  EditText txt;
  
  static
  {
    System.loadLibrary("native-lib");
  }
  
  public native boolean checkFlag(String paramString);
  
  protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(2131296284);
    this.but = ((Button)findViewById(2131165218));
    this.txt = ((EditText)findViewById(2131165238));
    this.but.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View paramAnonymousView)
      {
        paramAnonymousView = MainActivity.this;
        if (paramAnonymousView.checkFlag(paramAnonymousView.txt.getText().toString())) {
          Toast.makeText(MainActivity.this, "right!", 0).show();
        } else {
          Toast.makeText(MainActivity.this, "wrong!", 0).show();
        }
      }
    });
  }
}
```

这里只是声明了一下checkflag函数 要去native-lib里找

分析\glass\Project\lib\armeabi-v7a中的libnative-lib.so

IDA打开

怀疑是RC4加密 参考 https://www.cnblogs.com/SunsetR/p/12247041.html

```
bool __fastcall Java_com_ciscn_glass_MainActivity_checkFlag(int a1, int a2, int a3)
{
  const char *v3; // r4
  size_t v4; // r5
  int v6; // [sp+0h] [bp-220h]
  char v7; // [sp+100h] [bp-120h]

  v3 = (const char *)sub_F0C(a1, a3);
  if ( strlen(v3) != 39 )                    //flag长度为39
    return 0;
  _aeabi_memclr8(&v7, 256);
  _aeabi_memcpy8(&v6, "12345678", 256);  //定义v6 v7  给v6赋值12345678
  v4 = strlen((const char *)&v6);       //v4为v6的长度 8
  sub_FFC(&v7, &v6, v4);     //RC4填充S盒，s盒乱序
  sub_1088(&v7, v3, 39);     //RC4计算密钥流，异或加密
  sub_10D4(v3, 39, &v6, v4);    //自己搞得加密
  return memcmp(v3, &unk_497C, 0x27u) == 0;
}
```

```
unk_497C
0xa3, 0x1a, 0xe3, 0x69, 0x2f, 0xbb, 0x1a, 0x84, 0x65, 0xc2, 0xad, 0xad, 0x9e, 0x96, 0x5, 0x2, 0x1f, 0x8e, 0x36,
0x4f, 0xe1, 0xeb, 0xaf, 0xf0, 0xea, 0xc4, 0xa8, 0x2d, 0x42, 0xc7, 0x6e, 0x3f, 0xb0, 0xd3, 0xcc, 0x78, 0xf9, 0x98
, 0x3f
```

```
 sub_10D4(v3, 39, &v6, v4);
 int __fastcall sub_10D4(int result, int a2, int a3, int a4)
{
  int i; // r4
  int v5; // r6
  char v6; // r5
  char v7; // lr
  char v8; // r12
  int j; // lr
  int k; // r6

  for ( i = 0; i < a2; i += 3 )
  {
    v5 = result + i;
    v6 = *(_BYTE *)(result + i + 2);
    v7 = *(_BYTE *)(result + i + 1);
    v8 = *(_BYTE *)(result + i) ^ v6;
    *(_BYTE *)(result + i) = v8;
    *(_BYTE *)(v5 + 2) = v6 ^ v7;
    *(_BYTE *)(v5 + 1) = v7 ^ v8;   //交换异或
  }
  for ( j = 0; j < a2; j += a4 )
  {
    for ( k = 0; (a4 & ~(a4 >> 31)) != k && j + k < a2; ++k )
      *(_BYTE *)(result + k) ^= *(_BYTE *)(a3 + k); //分别异或12345678
    result += a4;
  }
  return result;
}
```

解题代码如下

```
#include <stdio.h>
#include <windows.h>
```

```
unsigned char *base64_encode(unsigned char *str)
{
    long len;
    long str_len;
    unsigned char *res;
    int i,j;
//定义base64编码表
    unsigned char *base64_table="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

//计算经过base64编码后的字符串长度
    str_len=strlen(str);
    if(str_len % 3 == 0)
        len=str_len/3*4;
    else
        len=(str_len/3+1)*4;

    res=malloc(sizeof(unsigned char)*len+1);
    res[len]='\0';

//以3个8位字符为一组进行编码
    for(i=0,j=0;i<len-2;j+=3,i+=4) {="" res[i]="base64_table[str[j]">>2]; //取出第一个字符的前6位并找出对应的结果字符
        res[i+1]=base64_table[(str[j]&0x3)<<4 | (str[j+1]>>4)]; //将第一个字符的后位与第二个字符的前4位进行组合并找到
对应的结果字符
        res[i+2]=base64_table[(str[j+1]&0xf)<<2 | (str[j+2]>>6)]; //将第二个字符的后4位与第三个字符的前2位组合并找出对
应的结果字符
        res[i+3]=base64_table[str[j+2]&0x3f]; //取出第三个字符的后6位并找出结果字符
    }

    switch(str_len % 3)
    {
        case 1:
            res[i-2]='=';
            res[i-1]='=';
            break;
        case 2:
            res[i-1]='=';
            break;
    }

    return res;
}

int main(){
 int j=0,i=0;

    byte result[]  = { 0xa3, 0x1a, 0xe3, 0x69, 0x2f, 0xbb, 0x1a,  0x84, 0x65,0xc2,0xad, 0xad,  0x9e, 0x96, 0x5,
0x2, 0x1f, 0x8e, 0x36, 0x4f,  0xe1,  0xeb,0xaf,  0xf0,  0xea,  0xc4,  0xa8, 0x2d, 0x42,  0xc7, 0x6e, 0x3f, 0xb0,
  0xd3,0xcc, 0x78,  0xf9,  0x98, 0x3f};
    char key[] = "1234567812345678123456781234567812345678";

    for (i = 0; i < sizeof(result); i++) {
        result[i] = result[i] ^ key[i];
    }
    for (j = 0; j < sizeof(result); j += 3) {
        result[j + 1] =  result[j + 1] ^ result[j];
        result[j + 2] = result[j + 2] ^ result[j + 1];
        result[j] = result[j] ^ result[j + 2];
    }
    char* flag = base64_encode(result);
    printf(flag);


    return 0;
}
```

+Lpql0fK6JHFB273kgs5khSor36qUEWNbS22hm6fhl7fsx5SpmJq

秘钥12345678

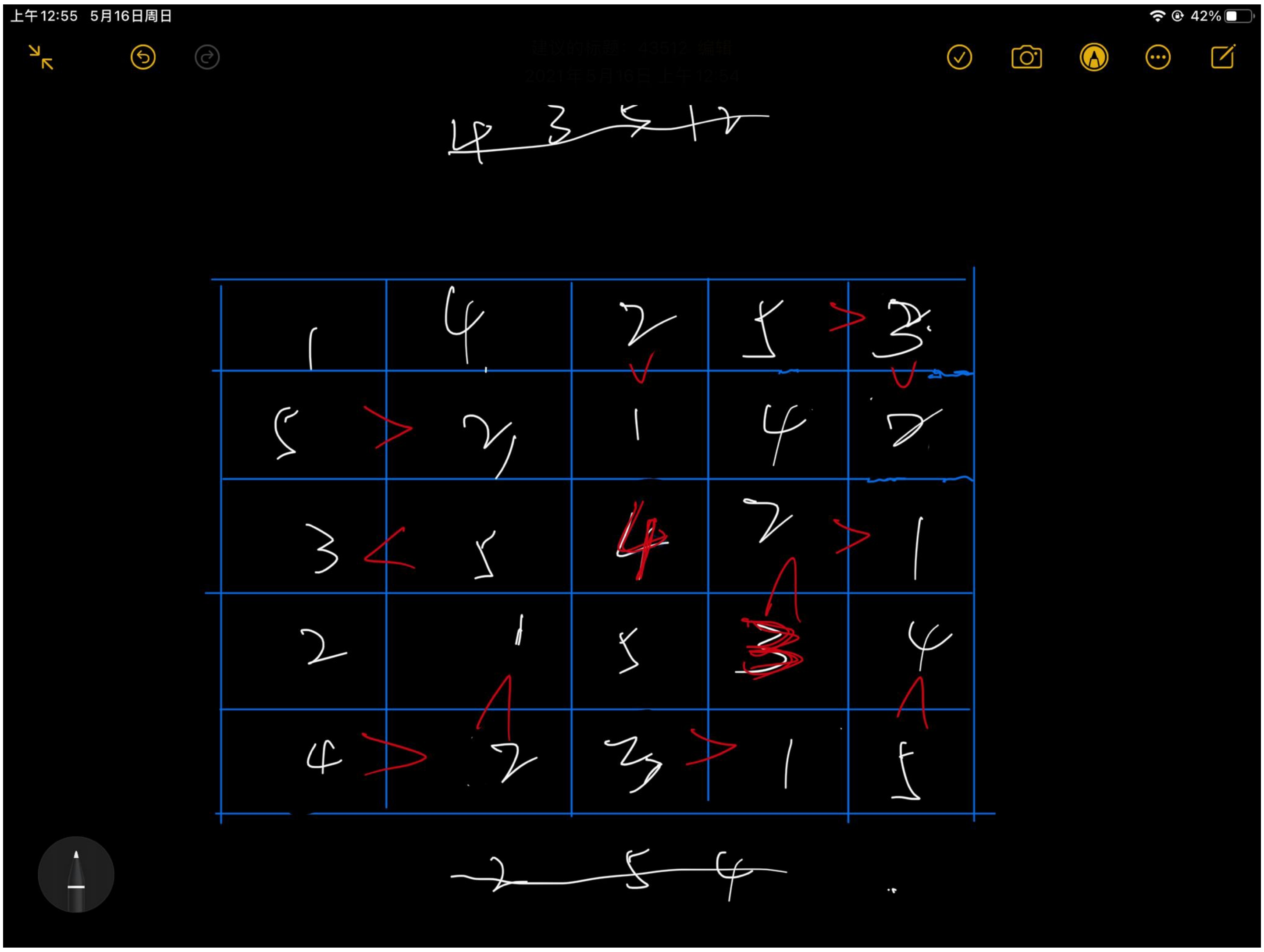
CISCN{6654d84617f627c88846c172e0f4d46c}

babybc

##方法一：

手撕出来的（惭愧）

有约束条件的数独 给出了两个数



拿到bc文件

根据https://stackoverflow.com/questions/32523847/how-to-make-llvm-bc-file-executable导出32位elf文件

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t v3; // rax
  unsigned __int64 v4; // rcx
  unsigned __int8 v5; // dl

  __isoc99_scanf(&unk_402004, x, envp);
  if ( (unsigned int)strlen(x) == 25 ) //25位flag
  {//数组所有的元素都<=5
    if ( x[0] )
    {
      if ( (unsigned __int8)(x[0] - 48) > 5u )
        return 0;
      v3 = strlen(x);
      v4 = 1LL;
      while ( v4 < v3 )
      {
        v5 = x[v4++] - 48;//chr数字转int数字
        if ( v5 > 5u )
          return 0;
      }
    }
    if ( f((__int64)x) && c() )//两个check函数都得为1
      printf("CISCN{MD5(%s)}", x);
  }
  return 0;
}
```

```c
char __fastcall f(__int64 a1)
{
  __int64 v1; // rax
  __int64 i; // rcx
  char v3; // dl
  char v4; // dl
  char v5; // dl
  char v6; // dl
  char v7; // dl
//填充数字 m[]为0则填充 不为0则判断输入的数是否为0,不为0报错 所以要让m[]中不为0位置的输入为0
    //这个函数的作用就是 让m[]中的4 3 位置输入为0
  v1 = 4LL;
  for ( i = 0LL; i < 5; ++i )
  {
    v3 = *(_BYTE *)(a1 + v1 - 4);
    if ( m[v1 - 4] )
    {
      if ( v3 != 48 )
        return 0;
    }
    else
    {
      if ( v3 == 48 )
        return 0;
      m[v1 - 4] = v3 - 48;
    }
    v4 = *(_BYTE *)(a1 + v1 - 3);
    if ( m[v1 - 3] )
    {
      if ( v4 != 48 )
        return 0;
    }
    }
    else
    {
      if ( v4 == 48 )
        return 0;
      m[v1 - 3] = v4 - 48;
    }
    v5 = *(_BYTE *)(a1 + v1 - 2);
    if ( m[v1 - 2] )
    {
      if ( v5 != 48 )
        return 0;
    }
    else
    {
      if ( v5 == 48 )
        return 0;
      m[v1 - 2] = v5 - 48;
    }
    v6 = *(_BYTE *)(a1 + v1 - 1);
    if ( m[v1 - 1] )
    {
      if ( v6 != 48 )
        return 0;
    }
    else
    {
      if ( v6 == 48 )
        return 0;
      m[v1 - 1] = v6 - 48;
    }
    v7 = *(_BYTE *)(a1 + v1);
    if ( m[v1] )
    {
      if ( v7 != 48 )
        return 0;
    }
    else
    {
      if ( v7 == 48 )
        return 0;
      m[v1] = v7 - 48;
    }
    v1 += 5LL;
  }
  return 1;
}
```

```
char c()
{
  unsigned __int8 *v0; // rax
  __int64 v1; // rdx
  __int64 v2; // rsi
  __int64 v3; // rsi
  __int64 v4; // rsi
  __int64 v5; // rsi
  __int64 v6; // rax
  __int64 v7; // rdx
  __int64 v8; // rdx
  __int64 v9; // rdx
```

```
__int64 v10; // rdx
unsigned __int8 *v11; // rax
__int64 v12; // rdx
char v13; // cl
char v14; // cl
char v15; // cl
char v16; // cl
__int64 v17; // rdx
__int64 v18; // rsi
char result; // al
char v20; // al
char v21; // al
char v22; // al
char v23; // al
char v24; // al
int v25; // [rsp+0h] [rbp-10h]
__int16 v26; // [rsp+4h] [rbp-Ch]
int v27; // [rsp+8h] [rbp-8h]
__int16 v28; // [rsp+Ch] [rbp-4h]

v0 = &m[4];
v1 = 0LL;
while ( 1 )
{
  v28 = 0;
  v27 = 0;
  v2 = *(v0 - 4);
  if ( *((_BYTE *)&v27 + v2) )
    break;
  *((_BYTE *)&v27 + v2) = 1;
  v3 = *(v0 - 3);
  if ( *((_BYTE *)&v27 + v3) )
    break;
  *((_BYTE *)&v27 + v3) = 1;
  v4 = *(v0 - 2);
  if ( *((_BYTE *)&v27 + v4) )
    break;
  *((_BYTE *)&v27 + v4) = 1;
  v5 = *(v0 - 1);
  if ( *((_BYTE *)&v27 + v5) )
    break;
  *((_BYTE *)&v27 + v5) = 1;
  if ( *((_BYTE *)&v27 + *v0) )
    break;
  ++v1;
  v0 += 5;
  if ( v1 >= 5 )
  {
    v6 = 0LL;
    while ( 1 )
    {
      v26 = 0;
      v25 = 0;
      v7 = m[v6];
      if ( *((_BYTE *)&v25 + v7) )
        break;
      *((_BYTE *)&v25 + v7) = 1;
      v8 = m[v6 + 5];
      if ( *((_BYTE *)&v25 + v8) )
```

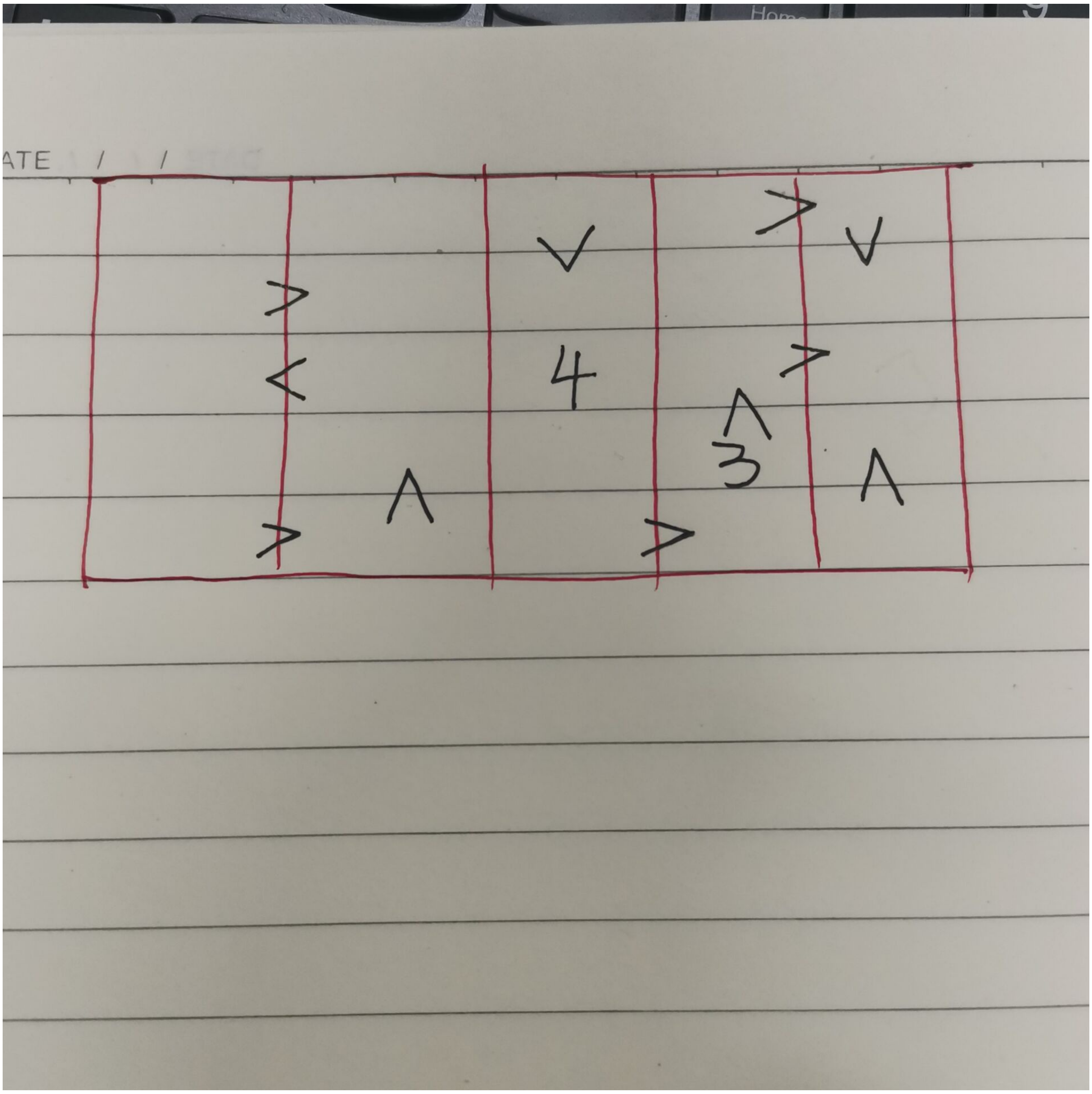
```
          break;
        *((_BYTE *)&v25 + v8) = 1;
        v9 = m[v6 + 10];
        if ( *((_BYTE *)&v25 + v9) )
          break;
        *((_BYTE *)&v25 + v9) = 1;
        v10 = m[v6 + 15];
        if ( *((_BYTE *)&v25 + v10) )
          break;
        *((_BYTE *)&v25 + v10) = 1;
        if ( *((_BYTE *)&v25 + m[v6 + 20]) )
          break;
//关键代码在这 两个循环分别代表行 和 列
//n是行 o是列 m是整个地图

        if ( ++v6 >= 5 )
        {
          v11 = &m[4];
          v12 = 0LL;
          //4个判断 因为5*5的棋盘 两两比较 只需要四次
              /*n[]={0,0,0,1
                 1,0,0,0
                 2,0,0,1
                 0,0,0,0
                 1,0,1,0 有用的就到这
                 0,0,0,0
                 0,0,0,0
                 0,0,0,0}
              */
         //1 左大 2 右大
          while ( 1 ) {
            v13 = n[4 * v12];
            if ( v13 == 2 )
            {
              if ( *(v11 - 4) > *(v11 - 3) )
                return 0;
            }
            else if ( v13 == 1 && *(v11 - 4) < *(v11 - 3) )
            {
              return 0;
            }
            v14 = n[4 * v12 + 1];
            if ( v14 == 1 )
            {
              if ( *(v11 - 3) < *(v11 - 2) )
                return 0;
            }
            else if ( v14 == 2 && *(v11 - 3) > *(v11 - 2) )
            {
              return 0;
            }
            v15 = n[4 * v12 + 2];
            if ( v15 == 2 )
            {
              if ( *(v11 - 2) > *(v11 - 1) )
                return 0;
            }
            else if ( v15 == 1 && *(v11 - 2) < *(v11 - 1) )
            {
              return 0;
```

```
}
v16 = n[4 * v12 + 3];
if ( v16 == 2 )
{
  if ( *(v11 - 1) > *v11 )
    return 0;
}
else if ( v16 == 1 && *(v11 - 1) < *v11 )
{
  return 0;
}
++v12;
v11 += 5;
if ( v12 >= 5 )
{
  v17 = 4LL;
  v18 = 0LL;
    /*o[]表示列
    o[]={0,0,2,0
     2,0,0,0
     0,0,0,0
     0,1,0,0
     1,0,0,1}
     1 上大 2 下大
        每列的12 23 34 45分别比较
    */
  while ( 1 )
  {
    v20 = o[v17 - 4];
    if ( v20 == 2 )
    {
      if ( m[v17 - 4] < m[v17 + 1] )
        return 0;
    }
    else if ( v20 == 1 && m[v17 - 4] > m[v17 + 1] )
    {
      return 0;
    }
    v21 = o[v17 - 3];
    if ( v21 == 1 )
    {
      if ( m[v17 - 3] > m[v17 + 2] )
        return 0;
    }
    else if ( v21 == 2 && m[v17 - 3] < m[v17 + 2] )
    {
      return 0;
    }
    v22 = o[v17 - 2];
    if ( v22 == 2 )
    {
      if ( m[v17 - 2] < m[v17 + 3] )
        return 0;
    }
    else if ( v22 == 1 && m[v17 - 2] > m[v17 + 3] )
    {
      return 0;
    }
    v23 = o[v17 - 1];
```

```
                if ( v23 == 2 )
                {
                  if ( m[v17 - 1] < m[v17 + 4] )
                    return 0;
                }
                else if ( v23 == 1 && m[v17 - 1] > m[v17 + 4] )
                {
                  return 0;
                }
                v24 = o[v17];
                if ( v24 == 2 )
                {
                  if ( m[v17] < m[v17 + 5] )
                    return 0;
                }
                else if ( v24 == 1 && m[v17] > m[v17 + 5] )
                {
                  return 0;
                }
                ++v18;
                v17 += 5LL;
                result = 1;
                if ( v18 >= 4 )
                  return result;
              }
            }
          }
        }
      }
      return 0;
    }
  }
  return 0;
}
```

根据所得条件 画出map图

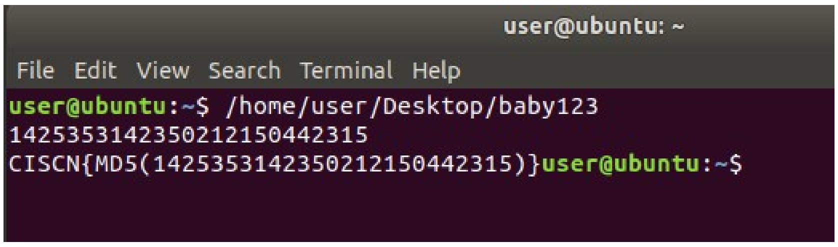


每行和每列都得有 12345并且不能有重复

数独做出来后为

14253 53142 35021 21504 42315

测试一下



```
user@ubuntu:~$ /home/user/Desktop/baby123
1425353142350212150442315
CISCN{MD5(1425353142350212150442315)}user@ubuntu:~$
```

md5后CISCN{8a04b4597ad08b83211d3adfa1f61431}

##方法二：
下载文件后,发现是.bc文件,使用命令编译成可执行文件:
llc -filetype=obj baby.bc
gcc baby.o
使用ida打开,发现主函数:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t v3; // rax
  unsigned __int64 v4; // rcx
  unsigned __int8 v5; // dl

  __isoc99_scanf(&unk_402004, x, envp);
  if ( (unsigned int)strlen(x) == 25 )
  {
    if ( x[0] )
    {
      if ( (unsigned __int8)(x[0] - 48) > 5u )
        return 0;
      v3 = strlen(x);
      v4 = 1LL;
      while ( v4 < v3 )
      {
        v5 = x[v4++] - 48;
        if ( v5 > 5u )
          return 0;
      }
    }
    if ( f((__int64)x) && c() )
      printf("CISCN{MD5(%s)}", x);
  }
  return 0;
}
```

分析主函数可知:输入的长度为25位,并且每位都不大于5,最终flag为输入值的md5值外面加上CISCN{},经过f((int64)x) 和c()两个函数校验之后确定输入的值是不是flag.查看这两个函数:f((int64)x):

```
char __fastcall f(__int64 a1)
{
  signed __int64 v1; // rax
  signed __int64 v2; // rcx
  char v3; // dl
  char v4; // dl
  char v5; // dl
  char v6; // dl
  char v7; // dl

  v1 = 4LL;
  v2 = 0LL;
  do
  {
    v3 = *(_BYTE *)(a1 + v1 - 4);
    if ( m[v1 - 4] )
    {
      if ( v3 != 48 )
        return 0;
    }
    else
    {
      if ( v3 == 48 )
        return 0;
      m[v1 - 4] = v3 - 48;
    }
    v4 = *(_BYTE *)(a1 + v1 - 3);
    if ( m[v1 - 3] )
    {
      if ( v4 != 48 )
        return 0;
    }
    else
    {
      if ( v4 == 48 )
        return 0;
      m[v1 - 3] = v4 - 48;
    }
    v5 = *(_BYTE *)(a1 + v1 - 2);
    if ( m[v1 - 2] )
    {
      if ( v5 != 48 )
        return 0;
    }
    else
    {
      if ( v5 == 48 )
        return 0;
      m[v1 - 2] = v5 - 48;
    }
    v6 = *(_BYTE *)(a1 + v1 - 1);
    if ( m[v1 - 1] )
    {
      if ( v6 != 48 )
        return 0;
    }
    else
    {
      if ( v6 == 48 )
        return 0;
      m[v1 - 1] = v6 - 48;
    }
    v7 = *(_BYTE *)(a1 + v1);
    if ( m[v1] )
    {
      if ( v7 != 48 )
```

```
        return 0;
    }
    else
    {
      if ( v7 == 48 )
        return 0;
      m[v1] = v7 - 48;
    }
    ++v2;
    v1 += 5LL;
  }
  while ( v2 < 5 );
  return 1;
}
```

c():

```

char c()
{
  unsigned __int8 *v0; // rax
  signed __int64 v1; // rdx
  __int64 v2; // rsi
  __int64 v3; // rsi
  __int64 v4; // rsi
  __int64 v5; // rsi
  __int64 v6; // rax
  __int64 v7; // rdx
  __int64 v8; // rdx
  __int64 v9; // rdx
  __int64 v10; // rdx
  unsigned __int8 *v11; // rax
  signed __int64 v12; // rdx
  char v13; // cl
  char v14; // cl
  char v15; // cl
  char v16; // cl
  signed __int64 v17; // rdx
  signed __int64 v18; // rsi
  char result; // al
  char v20; // al
  char v21; // al
  char v22; // al
  char v23; // al
  char v24; // al
  int v25; // [rsp+0h] [rbp-10h]
  __int16 v26; // [rsp+4h] [rbp-Ch]
  int v27; // [rsp+8h] [rbp-8h]
  __int16 v28; // [rsp+Ch] [rbp-4h]

  v0 = &m[4];
  v1 = 0LL;
  while ( 1 )
  {
    v28 = 0;
    v27 = 0;
    v2 = *(v0 - 4);
    if ( *((_BYTE *)&v27 + v2) )
      break;
    *((_BYTE *)&v27 + v2) = 1;
```

```
*((_BYTE *)&v27 + v2) = 1;
v3 = *(v0 - 3);
if ( *((_BYTE *)&v27 + v3) )
  break;
*((_BYTE *)&v27 + v3) = 1;
v4 = *(v0 - 2);
if ( *((_BYTE *)&v27 + v4) )
  break;
*((_BYTE *)&v27 + v4) = 1;
v5 = *(v0 - 1);
if ( *((_BYTE *)&v27 + v5) )
  break;
*((_BYTE *)&v27 + v5) = 1;
if ( *((_BYTE *)&v27 + *v0) )
  break;
++v1;
v0 += 5;
if ( v1 >= 5 )
{
  v6 = 0LL;
  while ( 1 )
  {
    v26 = 0;
    v25 = 0;
    v7 = m[v6];
    if ( *((_BYTE *)&v25 + v7) )
      break;
    *((_BYTE *)&v25 + v7) = 1;
    v8 = m[v6 + 5];
    if ( *((_BYTE *)&v25 + v8) )
      break;
    *((_BYTE *)&v25 + v8) = 1;
    v9 = m[v6 + 10];
    if ( *((_BYTE *)&v25 + v9) )
      break;
    *((_BYTE *)&v25 + v9) = 1;
    v10 = m[v6 + 15];
    if ( *((_BYTE *)&v25 + v10) )
      break;
    *((_BYTE *)&v25 + v10) = 1;
    if ( *((_BYTE *)&v25 + m[v6 + 20]) )
      break;
    if ( ++v6 >= 5 )
    {
      v11 = &m[4];
      v12 = 0LL;
      while ( 1 )
      {
        v13 = n[4 * v12];
        if ( v13 == 2 )
        {
          if ( *(v11 - 4) > *(v11 - 3) )
            return 0;
        }
        else if ( v13 == 1 && *(v11 - 4) < *(v11 - 3) )
        {
          return 0;
        }
        v14 = n[4 * v12 + 1];
        if ( v14 == 1 )
```

```
          {
            if ( *(v11 - 3) < *(v11 - 2) )
              return 0;
          }
          else if ( v14 == 2 && *(v11 - 3) > *(v11 - 2) )
          {
            return 0;
          }
          v15 = n[4 * v12 + 2];
          if ( v15 == 2 )
          {
            if ( *(v11 - 2) > *(v11 - 1) )
              return 0;
          }
          else if ( v15 == 1 && *(v11 - 2) < *(v11 - 1) )
          {
            return 0;
          }
          v16 = n[4 * v12 + 3];
          if ( v16 == 2 )
          {
            if ( *(v11 - 1) > *v11 )
              return 0;
          }
          else if ( v16 == 1 && *(v11 - 1) < *v11 )
          {
            return 0;
          }
          ++v12;
          v11 += 5;
          if ( v12 >= 5 )
          {
            v17 = 4LL;
            v18 = 0LL;
            while ( 1 )
            {
              v20 = o[v17 - 4];
              if ( v20 == 2 )
              {
                if ( m[v17 - 4] < m[v17 + 1] )
                  return 0;
              }
              else if ( v20 == 1 && m[v17 - 4] > m[v17 + 1] )
              {
                return 0;
              }
              v21 = o[v17 - 3];
              if ( v21 == 1 )
              {
                if ( m[v17 - 3] > m[v17 + 2] )
                  return 0;
              }
              else if ( v21 == 2 && m[v17 - 3] < m[v17 + 2] )
              {
                return 0;
              }
              v22 = o[v17 - 2];
              if ( v22 == 2 )
              {
                if ( m[v17 - 2] < m[v17 + 3] )
```

```
                  if ( m[v17 - 2] < m[v17 + 3] )
                    return 0;
                }
                else if ( v22 == 1 && m[v17 - 2] > m[v17 + 3] )
                {
                  return 0;
                }
                v23 = o[v17 - 1];
                if ( v23 == 2 )
                {
                  if ( m[v17 - 1] < m[v17 + 4] )
                    return 0;
                }
                else if ( v23 == 1 && m[v17 - 1] > m[v17 + 4] )
                {
                  return 0;
                }
                v24 = o[v17];
                if ( v24 == 2 )
                {
                  if ( m[v17] < m[v17 + 5] )
                    return 0;
                }
                else if ( v24 == 1 && m[v17] > m[v17 + 5] )
                {
                  return 0;
                }
                ++v18;
                v17 += 5LL;
                result = 1;
                if ( v18 >= 4 )
                  return result;
              }
            }
          }
        }
      }
      return 0;
    }
  }
  return 0;
}
```

可以看出f(x)函数是先对m数组的值判断,如果当前值不是0,判断输入的值是不是0,如果不是,则退出,即代表输入错误,如果当前m值是0,判断输入值是0,则退出,代表输入错误,如果不是,则把输入值赋值给m数组当前值.可以得出输入的第13和第19位为0c()函数是对赋值后的m数组校验,先是判断相邻数字横向大小,再判断竖向,可以据此写出z3脚本求解:

```
from z3 import *
x1 = Int('x1')
x2 = Int('x2')
x3 = Int('x3')
x4 = Int('x4')
x5 = Int('x5')
x6 = Int('x6')
x7 = Int('x7')
x8 = Int('x8')
x9 = Int('x9')
x10 = Int('x10')
x11 = Int('x11')
```

```
x11 = Int('x11')
x12 = Int('x12')
x13 = Int('x13')
x14 = Int('x14')
x15 = Int('x15')
x16 = Int('x16')
x17 = Int('x17')
x18 = Int('x18')
x19 = Int('x19')
x20 = Int('x20')
x21 = Int('x21')
x22 = Int('x22')
x23 = Int('x23')
x24 = Int('x24')
x25 = Int('x25')
s = Solver()
s.add(x13 == 4,x19 == 3)
s.add(x1 < 6, x1 > 0)
s.add(x2 < 6, x2 > 0)
s.add(x3 < 6, x3 > 0)
s.add(x4 < 6, x4 > 0)
s.add(x5 < 6, x5 > 0)
s.add(x6 < 6, x6 > 0)
s.add(x7 < 6, x7 > 0)
s.add(x8 < 6, x8 > 0)
s.add(x9 < 6, x9 > 0)
s.add(x10 < 6, x10 > 0)
s.add(x11 < 6, x11 > 0)
s.add(x12 < 6, x12 > 0)
s.add(x13 < 6, x13 > 0)
s.add(x14 < 6, x14 > 0)
s.add(x15 < 6, x15 > 0)
s.add(x16 < 6, x16 > 0)
s.add(x17 < 6, x17 > 0)
s.add(x18 < 6, x18 > 0)
s.add(x19 < 6, x19 > 0)
s.add(x20 < 6, x20 > 0)
s.add(x21 < 6, x21 > 0)
s.add(x22 < 6, x22 > 0)
s.add(x23 < 6, x23 > 0)
s.add(x24 < 6, x24 > 0)
s.add(x25 < 6, x25 > 0)
s.add(x6 > x7)
s.add(x3 > x8)
s.add(x4 > x5)
s.add(x5 > x10)
s.add(x14 > x15)
s.add(x19 > x14)
s.add(x23 > x24)
s.add(x25 > x20)
s.add(x21 > x22)
s.add(x22 > x17)
s.add(x12 > x11)
s.add(x1 != x2, x1 != x3, x1 != x4, x1 != x5, x2 != x3, x2 != x4, x2 != x5, x3 != x4, x3 != x5, x4 != x5)
s.add(x6 != x7, x6 != x8, x6 != x9, x6 != x10, x7 != x8, x7 != x9, x7 != x10, x8 != x9, x8 != x10, x9 != x10)
s.add(x11 != x12, x11 != x13, x11 != x14, x11 != x15, x12 != x13, x21 != x14, x12 != x15, x13 != x14, x13 != x15
,
      x14 != x15)
s.add(x16 != x17, x16 != x18, x16 != x19, x16 != x20, x17 != x18, x17 != x19, x17 != x20, x18 != x19, x18 != x20
,
      x19 != x20)
s.add(x21 != x22, x21 != x23, x21 != x24, x21 != x25, x22 != x23, x22 != x24, x22 != x25, x23 != x24, x23 != x25
,
      x24 != x25)
s.add(x1 != x6, x1 != x11, x1 != x16, x1 != x21, x6 != x11, x6 != x16, x6 != x21, x11 != x16, x11 != x21, x16 !=
 x21)
s.add(x2 != x7, x2 != x12, x2 != x17, x2 != x22, x7 != x12, x7 != x17, x7 != x22, x12 != x17, x12 != x22, x17 !=
 x22)
s.add(x3 != x8, x3 != x13, x3 != x18, x3 != x23, x8 != x13, x8 != x18, x8 != x23, x13 != x18, x13 != x23, x18 !=
 x23)
s.add(x4 != x9, x4 != x14, x4 != x19, x4 != x24, x9 != x14, x9 != x19, x9 != x24, x14 != x19, x14 != x24, x19 !=
 x24)
s.add(x5 != x10, x5 != x15, x5 != x20, x5 != x25, x10 != x15, x10 != x20, x10 != x25, x15 != x20, x15 != x25,
      x20 != x25)
if s.check() == sat:
    m = s.model()
    print(m.eval(x1),m.eval(x2),m.eval(x3),m.eval(x4),
          m.eval(x5),m.eval(x6),m.eval(x7),m.eval(x8),
          m.eval(x9),m.eval(x10),m.eval(x11),m.eval(x12),
          m.eval(x13),m.eval(x14),m.eval(x15),m.eval(x16),
          m.eval(x17),m.eval(x18),m.eval(x19),m.eval(x20),
          m.eval(x21),m.eval(x22),m.eval(x23),m.eval(x24),m.eval(x25))
```

得到m数组值:1 4 2 5 3 5 3 1 4 2 3 5 4 2 1 2 1 5 3 4 4 2 3 1 5

则输入值为: 1425353142350212150442315