逃离 FDU：

```
if ( *v15 != 70 )
  return 0;
v16 = v15 + 1;
if ( *v16 != 76 )
  return 0;
v17 = v16 + 1;
if ( *v17 != 65 )
  return 0;
v18 = v17 + 1;
if ( *v18 != 71 )
  return 0;
v19 = v18 + 1;
if ( *v19 == 123 )
  v21 = check2(v19 + 1) & 1;
else
  v21 = 0;
return v21;
```

这一段是主要的 check2 的使用位置，内部字符串检查

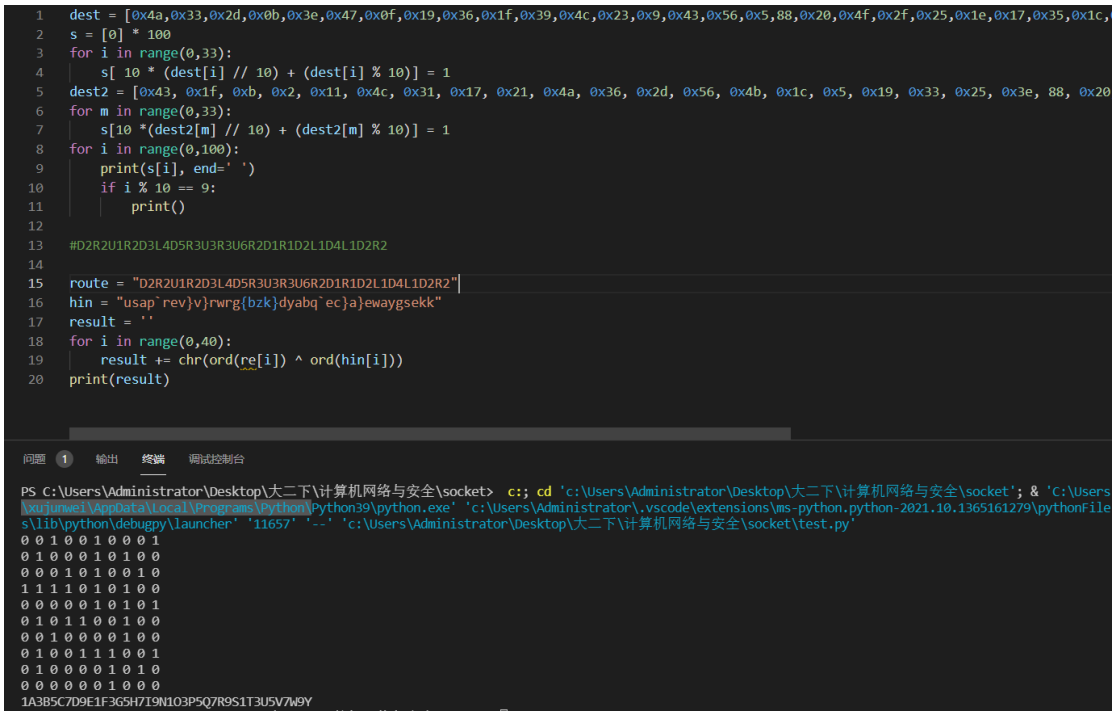
```
for ( i = 0; i < 33; ++i )
  *((_DWORD *)&s[5 * (dest[i] / 10)] + dest[i] % 10) = 1;
sub_32BF0(v21);
for ( j = 0; j < 33; ++j )
{
  if ( *((_DWORD *)&s[5 * (dest[j] % 10)] + dest[j] / 10) == 1 )
    sub_32C20(v21, &dest[j]);
}
v14 = 0;
v13 = 0;
v15 = 999990;
v16 = 999990;
for ( k = 2; k <= 7; ++k )
{
  v23[2 * k + 33] = v23[2 * v17 + 33] + 1;
  v23[2 * v17 + 34] = 1;
}
for ( l = 2; l <= 7; ++l )
{
  if ( v23[2 * l + 34] != 1 && v23[2 * l + 33] < v15 )
  {
    if ( v23[2 * l + 33] >= v16 )
    {
      v15 = v23[2 * l + 33];
    }
    else
    {
      v15 = v16;
      v16 = v23[2 * l + 33];
    }
  }
}
memcpy(v23, &unk_AAB54, 0x84u);
for ( m = 0; m < 33; ++m )
  *((_DWORD *)&s[5 * (v23[m] / 10)] + v23[m] % 10) = 1;
```

上面这一段是迷宫创造的逻辑，为 1 的是障碍物，为 0 是可以走的位置

```
switch ( *a1 )
{
  case 'U':
    ++a1;
    for ( jj = 0; jj < *a1 - 48; ++jj )
      --v14;
    break;
  case 'D':
    ++a1;
    for ( kk = 0; kk < *a1 - 48; ++kk )
      ++v14;
    break;
  case 'L':
    ++a1;
    for ( ll = 0; ll < *a1 - 48; ++ll )
      --v13;
    break;
  case 'R':
    ++a1;
    for ( mm = 0; mm < *a1 - 48; ++mm )
      ++v13;
    break;
}
if ( v14 < 0 || v14 > 9 || v13 < 0 || v13 > 9 )
{
  v20 = 0;
  goto LABEL_69;
}
if ( *((_DWORD *)&s[5 * v14] + v13) == 1 )
{
  v20 = 0;
  goto LABEL_69;
}
if ( v14 == 9 && v13 == 9 )
  break;
```

暴露自己是迷宫的是这个位置，UDLR 是上下左右（拳皇的按键哈哈哈哈，所以直接反映了），而这个 48 是 0 的 ASCII 码，所以也很易见是走几步的意思，说明就是朝一个方向走几步。而且从最后的 v14 == 9 && v13 == 9 可以看出来 迷宫是 10*10 的……之前 s 迷宫的变量的 _QWORD s[50] 也可以直接改为 int s[100]来操作

最后解迷宫和解码的结果如下：



```
dest = [0x4a,0x33,0x2d,0x0b,0x3e,0x47,0x0f,0x19,0x36,0x1f,0x39,0x4c,0x23,0x9,0x43,0x56,0x5,0x88,0x20,0x4f,0x2f,0x25,0x1e,0x17,0x35,0x1c,
s = [0] * 100
for i in range(0,33):
    s[ 10 * (dest[i] // 10) + (dest[i] % 10)] = 1
dest2 = [0x43, 0x1f, 0xb, 0x2, 0x11, 0x4c, 0x31, 0x17, 0x21, 0x4a, 0x36, 0x2d, 0x56, 0x4b, 0x1c, 0x5, 0x19, 0x33, 0x25, 0x3e, 88, 0x20
for m in range(0,33):
    s[10 *(dest2[m] // 10) + (dest2[m] % 10)] = 1
for i in range(0,100):
    print(s[i], end=' ')
    if i % 10 == 9:
        print()

#D2R2U1R2D3L4D5R3U3R3U6R2D1R1D2L1D4L1D2R2

route = "D2R2U1R2D3L4D5R3U3R3U6R2D1R1D2L1D4L1D2R2"
hin = "usap`rev}v}rwrg{bzk}dyabq`ec}a}ewaygsekk"
result = ''
for i in range(0,40):
    result += chr(ord(re[i]) ^ ord(hin[i]))
print(result)
```

```
问题 1  输出  终端  调试控制台

PS C:\Users\Administrator\Desktop\大二下\计算机网络与安全\socket>  c:; cd 'c:\Users\Administrator\Desktop\大二下\计算机网络与安全\socket'; & 'C:\Users
\xujunwei\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Administrator\.vscode\extensions\ms-python.python-2021.10.1365161279\pythonFile
s\lib\python\debugpy\launcher' '11657' '--' 'c:\Users\Administrator\Desktop\大二下\计算机网络与安全\socket\test.py'
0 0 1 0 0 1 0 0 0 1
0 1 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 1 0
1 1 1 1 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0 1
0 1 0 1 1 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0
0 1 0 0 1 1 1 0 0 1
0 1 0 0 0 0 1 0 1 0
0 0 0 0 0 0 1 0 0 0
1A3B5C7D9E1F3G5H7I9N1O3P5Q7R9S1T3U5V7W9Y
```

四方客栈：

这个题我做了一部分，是对于 native code 的逆向

先把原数组找到

```
*(_QWORD *)&v17[14] = 14LL;
*(_QWORD *)&v17[12] = 0x70000000DLL;
*(_QWORD *)&v17[10] = 0x60000000BLL;
*(_QWORD *)&v17[8] = 0xA00000002LL;
*(_QWORD *)&v17[6] = 0xF0000000CLL;
*(_QWORD *)&v17[4] = 0x400000009LL;
*(_QWORD *)&v17[2] = 0x300000008LL;
*(_QWORD *)v17 = 0x100000005LL;                 // 数组:5,1,8,3,9,4,c,f,2,a,b,6,d,7,e,0
v16 = 15;
v15 = 1;
v3 = (*env)->GetStringLength(env, string);
v4 = (*env)->GetStringUTFChars(env, string, 0);
```

下面这一段是个循环判断条件，其实就是结果为 1，2，3，4，5，6，7，8，9，a,b,c,d,e,0

只是作者进行了控制流平坦化，解读起来费些功夫

```
  }
  while ( v3 );
  v8 = v15 != 0;
}
result = 0;
if ( v8 && !v17[15] )                   // HIDWORD(v24) == 0
{
  v10 = 0;
  for ( i = -14; ; ++i )
  {
    v12 = v10++;
    if ( v10 <= 0xE )                   //  v10 == 15就结束，<=没意义
      break;
LABEL_14:
    if ( v10 == 15 )
      return 1;
  }
  v13 = v17[v12];
  v14 = i;
  while ( v13 <= v17[v14 + 15] )        // v13 == v17[0] && v14 == i
  {
    if ( !++v14 )                       // ++v14 == 0
      goto LABEL_14;
  }
```

主要是看 mtfa 函数是在干啥：

```
case 15:
  if ( a2 == 14 || a2 == 11 )
  {
    v32 = v17_a3[15] ^ v17_a3[a2];          // 互换
    v17_a3[15] = v32;
    v33 = v17_a3[a2] ^ v32;
    v17_a3[a2] = v33;
    v17_a3[15] ^= v33;
    *v16_a = a2;
  }
  else
  {
```

拿出其中一个 case 进行举例，中间 if 体里面的代码是把两个位置的数交换一下，然后下一次的交换位置是其中一个位置是 a2

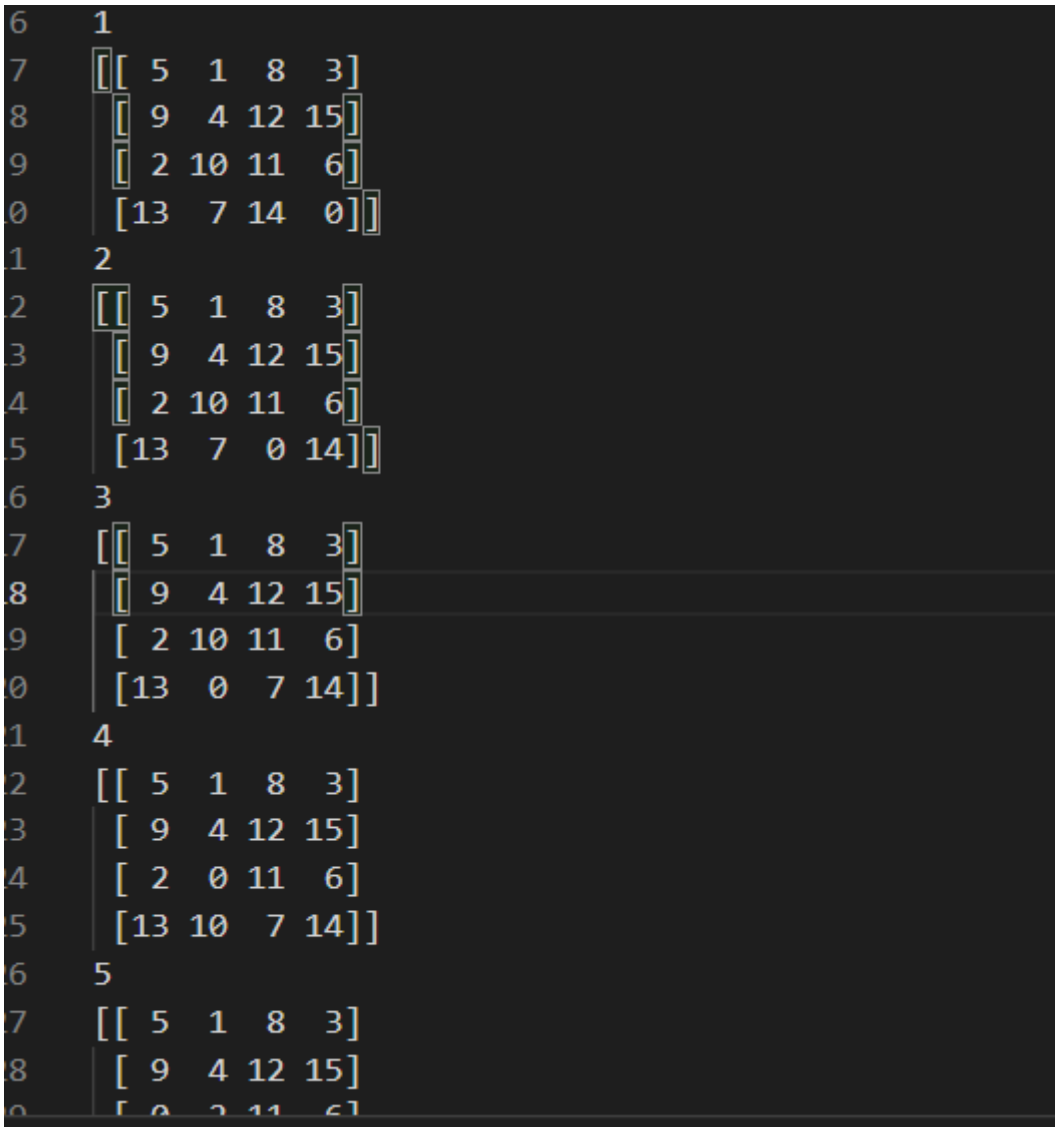
然而，15 的位置只能和 11，14 交换

再看看别的 case 的交换条件，比如 12 只能和 8，13 换

好多的限制条件，让我看出来是一个 4x4 的二维矩阵，然后 0 作为移动的标致，所以是拼图游戏（0 是空格位置）

然后就弄了个脚本

```python
def test():
    openlist = []    #open表
    close = []       #存储扩展的父节点

    print('请输入矩阵的行数')
    num = int(input())

    print("请输入初始矩阵A")
    A = get_input(num)

    print("请输入目标矩阵B")
    B = get_input(num)

    print("请输入结果文件名")
    resultfile = input()

    goal['vec'] = np.array(B)   #建立矩阵

    p = {}
    p['vec'] = np.array(A)
    p['dis'] = get_ManhattanDis(goal['vec'], p['vec'])
    p['step'] = 0
```



就能解出来这个拼图游戏的解然后交给了另外一个同学