In [10]:

```
import numpy as np
import math

# 二维DCT变换
def DCT_Transform(block):
    # 使范围保持在-127～128
    block = block -128
    res = np.zeros(block.shape)
    C = lambda x: math.sqrt(2)/2 if x==0 else 1
    for u in range(8):
            for v in range(8):
                sum = 0
                for i in range(8):
                    for j in range(8):
                        sum+=math.cos(math.pi*(2*i+1)*u/16)*math.cos(math.pi*v
*(2*j+1)/16)*(block[i][j])
                res[u,v] = round(sum *C(u)*C(v)/4)
    return res

# 书上例子用于测试
block = np.array([
                [200, 202, 189, 188, 189, 175, 175, 175],
                [200, 203, 198, 188, 189, 182, 178, 175],
                [203, 200, 200, 195, 200, 187, 185, 175],
                [200, 200, 200, 200, 197, 187, 187, 187],
                [200, 205, 200, 200, 195, 188, 187, 175],
                [200, 200, 200, 200, 200, 190, 187, 175],
                [205, 200, 199, 200, 191, 187, 187, 175],
                [210, 200, 200, 200, 188, 185, 187, 186]
            ])

dct_block = DCT_Transform(block)
print(dct_block)
```

```
[[ 515.   65.  -12.    4.    1.    2.   -8.    5.]
 [ -16.    3.    2.    0.   -0.  -11.   -2.    3.]
 [ -12.    6.   11.   -1.    3.    0.    1.   -2.]
 [  -8.    3.   -4.    2.   -2.   -3.   -5.   -2.]
 [   0.   -2.    7.   -5.    4.   -0.   -1.   -4.]
 [   0.   -3.   -1.    0.    4.    1.   -1.    0.]
 [   3.   -2.   -3.    3.    3.   -1.   -1.    3.]
 [  -2.    5.   -2.    4.   -2.    2.   -3.    0.]]
```

```
In [13]:
```

```
# 亮度和色度量化表
QY=np.array([[16,11,10,16,24,40,51,61],
             [12,12,14,19,26,48,60,55],
             [14,13,16,24,40,57,69,56],
             [14,17,22,29,51,87,80,62],
             [18,22,37,56,68,109,103,77],
             [24,35,55,64,81,104,113,92],
             [49,64,78,87,103,121,120,101],
             [72,92,95,98,112,100,103,99]])

QC=np.array([[17,18,24,47,99,99,99,99],
             [18,21,26,66,99,99,99,99],
             [24,26,56,99,99,99,99,99],
             [47,66,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99],
             [99,99,99,99,99,99,99,99]])

# 根据亮度量化表进行量化
quan_block = np.round(dct_block/QY)
print(quan_block)
```

```
[[ 32.   6.  -1.   0.   0.   0.  -0.   0.]
 [ -1.   0.   0.   0.  -0.  -0.  -0.   0.]
 [ -1.   0.   1.  -0.   0.   0.   0.  -0.]
 [ -1.   0.  -0.   0.  -0.  -0.  -0.  -0.]
 [  0.  -0.   0.  -0.   0.  -0.  -0.  -0.]
 [  0.  -0.  -0.   0.   0.   0.  -0.   0.]
 [  0.  -0.  -0.   0.   0.  -0.  -0.   0.]
 [ -0.   0.  -0.   0.  -0.   0.  -0.   0.]]
```

In [14]:

```
def block_to_zigzag(block):
    return np.array([block[point] for point in zigzag_points(*block.shape)])

# 返回横纵坐标
def zigzag_points(rows, cols):
    # 方向选择
    UP, DOWN, RIGHT, LEFT, UP_RIGHT, DOWN_LEFT = range(6)

    # 移动坐标
    def move(direction, point):
        return {
            UP: lambda point: (point[0] - 1, point[1]),
            DOWN: lambda point: (point[0] + 1, point[1]),
            LEFT: lambda point: (point[0], point[1] - 1),
            RIGHT: lambda point: (point[0], point[1] + 1),
            UP_RIGHT: lambda point: move(UP, move(RIGHT, point)),
            DOWN_LEFT: lambda point: move(DOWN, move(LEFT, point))
        }[direction](point)

    # 判断是否在边界内
    def inbounds(point):
        return 0 <= point[0] < rows and 0 <= point[1] < cols

    # 左上角开始
    point = (0, 0)

    # True when moving up-right, False when moving down-left
    move_up = True

    for i in range(rows * cols):
        yield point
        if move_up:
            if inbounds(move(UP_RIGHT, point)):
                point = move(UP_RIGHT, point)
            else:
                move_up = False
                if inbounds(move(RIGHT, point)):
                    point = move(RIGHT, point)
                else:
                    point = move(DOWN, point)
        else:
            if inbounds(move(DOWN_LEFT, point)):
                point = move(DOWN_LEFT, point)
            else:
                move_up = True
                if inbounds(move(DOWN, point)):
                    point = move(DOWN, point)
                else:
                    point = move(RIGHT, point)

z_array = block_to_zigzag(quan_block)
print(z_array)
```

```
[ 32.   6.  -1.  -1.   0.  -1.   0.   0.   0.  -1.   0.   0.   1.
0.   0.
   0.  -0.  -0.  -0.  -0.   0.   0.  -0.   0.   0.   0.  -0.  -0.
0.  -0.
   0.  -0.  -0.  -0.  -0.  -0.   0.  -0.   0.   0.  -0.   0.   0.
-0.  -0.
  -0.   0.   0.  -0.   0.   0.   0.  -0.  -0.  -0.  -0.  -0.  -0.
0.  -0.
   0.   0.  -0.   0.]
```

In [26]:

```python
def reverse_str(str):
    res = ""
    for _,s in enumerate(str):
        if s=='0':
            res += '1'
        else:
            res += '0'
    return res

def binstr_flip(binstr):
    # check if binstr is a binary string
    if not set(binstr).issubset('01'):
        raise ValueError("binstr should have only '0's and '1's")
    return ''.join(map(lambda c: '0' if c == '1' else '1', binstr))

# 输入VALUE，得到SIZE
def int_size(number):
    str = ""
    if number==0:
        return 0
    if number < 0:
        number = abs(number)
        l = len(bin(number)) - 2
        str = bin(number)[-l:]
        str = reverse_str(str)
    else:
        l = len(bin(number)) - 2
        str = bin(number)[-l:]
    # return (l,str)
    return l

# 输入VALUE，得到幅值AMPLITUDE
def int_to_binstr(n):
    if n == 0:
        return ''

    binstr = bin(abs(n))[2:]

    # change every 0 to 1 and vice verse when n is negative
    return binstr if n > 0 else binstr_flip(binstr)

# 返回（RUNLENGTH，SIZE）和 “AMPLIYTUDE“
def run_length_encode(arr):
    # determine where the sequence is ending prematurely
    last_nonzero = -1
    # print(arr)

    for i, elem in enumerate(arr):
        if elem != 0:
            last_nonzero = i
    # print(last_nonzero)

    # each symbols1 is a (RUNLENGTH, SIZE) tuple
    # each symbols2 is a AMPLITUDE string
    symbols1 = []
    symbols2 = []
    run_length = 0

    for i, elem in enumerate(arr):
        if i > last_nonzero:
            symbols1.append((0, 0))
            symbols2.append(0)
            break
        elif elem == 0 and run_length < 15:
            run_length += 1
        else:
            # size = bits_required(elem)
            symbols1.append((run_length, int_size(elem)))
            symbols2.append(int_to_binstr(elem))
            run_length = 0
    return symbols1,symbols2

# 注意转换为32位整型
z_array = z_array.astype(np.int32)
# 注意从第二个下标开始进行游长编码
symbols1,symbols2 = run_length_encode(z_array[1:])
print("symbols1:(RUNLENGTH,SIZE)\n",symbols1)
print("symbols2:(AMPLITUDE)\n",symbols2)
```

```
symbols1:(RUNLENGTH,SIZE)
 [(0, 3), (0, 1), (0, 1), (1, 1), (3, 1), (2, 1), (0, 0)]
symbols2:(AMPLITUDE)
 ['110', '0', '0', '0', '0', '1', 0]
```

In [31]:

```
# 输入DC系数数组，输出SIZE和AMPLITUDE
def dcpm(dc_array):
    sizes = []
    amplitudes = []
    for i in range(dc_array.shape[0]):
        if i==0:
            sizes.append(int_size(dc_array[i]))
            amplitudes.append(int_to_binstr(dc_array[i]))
        else:
            sizes.append(int_size(dc_array[i]-dc_array[i-1]))
            amplitudes.append(int_to_binstr(dc_array[i]-dc_array[i-1]))
    return sizes,amplitudes

# 以书中例子作为测试
dc_arrays = np.array([150,155,149,152,144])
sizes,amplitudes = dcpm(dc_arrays)
print("SIZE:\n",sizes)
print("AMPLITUDE:\n",amplitudes)
```

```
SIZE:
 [8, 3, 3, 2, 4]
AMPLITUDE:
 ['10010110', '101', '001', '11', '0111']
```

In [32]:

```
from huffmantree import HuffmanTree

H_DC = HuffmanTree(sizes)
H_AC = HuffmanTree(symbols1)

print(H_DC.value_to_bitstring_table())
print(H_AC.value_to_bitstring_table())
```

```
---------------------------------------------------------------------------
---------
ModuleNotFoundError                       Traceback (most recent c
all last)
<ipython-input-32-69f5bb346970> in <module>()
----> 1 from huffmantree import HuffmanTree
      2 
      3 H_DC = HuffmanTree(sizes)
      4 H_AC = HuffmanTree(symbols1)
      5 

ModuleNotFoundError: No module named 'huffmantree'
```