三、编程题

一、算法的核心思想是**分治法**：

1. 如果当前子图像的所有像素点颜色相同，则直接返回结构度 1。
2. 如果当前子图像的像素点颜色不同，则将其分割为四个更小的子图像，递归计算每个子图像的结构度。
3. 最终，当前子图像的结构度等于四个更小子图像的结构度之和加 1。

这种递归分治的方式可以高效地处理大规模图像，避免了暴力计算的复杂度。

二、分析：

1. 递归终止条件

```
if size == 1:
    return 1
```

如果当前子图像的大小为 1×1（即只有一个像素点），根据定义，其结构度为 1。

这是递归的终止条件，防止无限递归。

2. 检查子图像是否所有像素相同

```
first_pixel = image[x][y]
all_same = True
for i in range(x, x + size):
    for j in range(y, y + size):
        if image[i][j] != first_pixel:
            all_same = False
            break
    if not all_same:
        break
```

从当前子图像的左上角像素点开始，检查所有像素点是否与第一个像素点相同。

如果发现任何一个像素点与第一个像素点不同，则 all_same 标志设为 False，并退出循环。

这一步是为了判断当前子图像是否满足“所有像素点颜色相同”的条件。

3. 处理所有像素相同的子图像

```
if all_same:
    return 1
```

如果当前子图像的所有像素点颜色相同，根据定义，其结构度为 1。

直接返回 1，不再进一步分割。

4. 分割子图像并递归计算

```
sub_size = size // 2
structure_degree = (
    calculate_structure_degree(image, x, y, sub_size) +
    calculate_structure_degree(image, x, y + sub_size, sub_size) +
    calculate_structure_degree(image, x + sub_size, y, sub_size) +
    calculate_structure_degree(image, x + sub_size, y + sub_size, sub_size)
)
```

如果当前子图像的像素点不完全相同，则将当前子图像分割为四个更小的子图像：

- 左上子图像：左上角为 (x, y)，大小为 sub_size x sub_size。
- 右上子图像：左上角为 (x, y + sub_size)，大小为 sub_size x sub_size。
- 左下子图像：左上角为 (x + sub_size, y)，大小为 sub_size x sub_size。

- 右下子图像：左上角为 (x + sub_size, y + sub_size)，大小为 sub_size x sub_size。

对这四个更小的子图像分别递归调用 calculate_structure_degree 函数，计算它们的结构度。

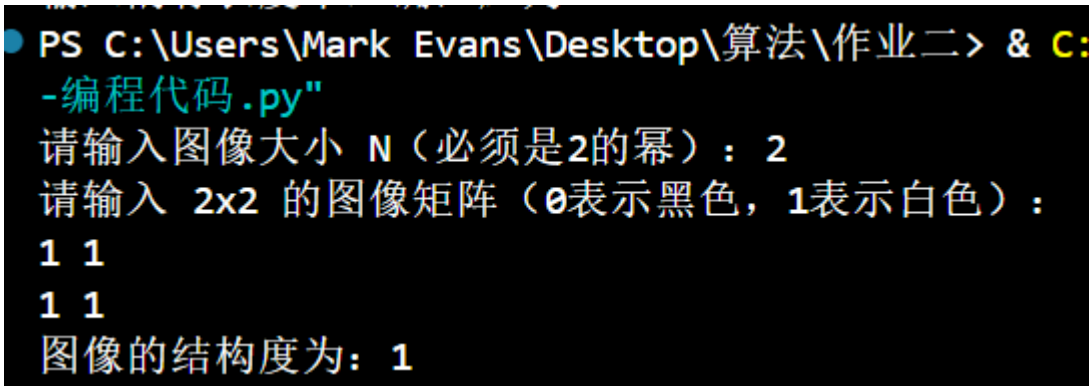
5. 计算当前子图像的结构度

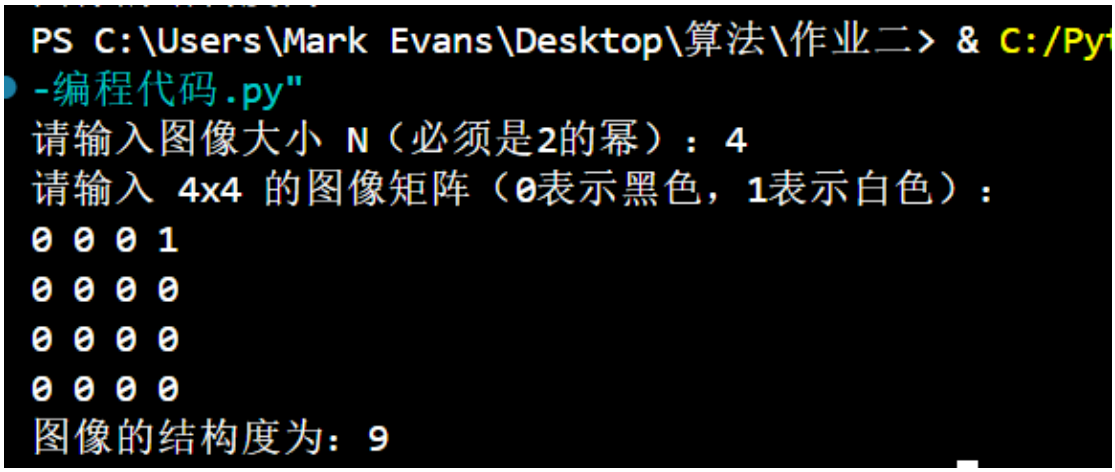
return structure_degree + 1

根据定义，当前子图像的结构度等于四个更小子图像的结构度之和，再加上 1。

这里的 +1 是因为当前子图像本身也需要计入结构度。

三、运行结果：





思考题：

(1)：

1. 分割图像为四个子图像：
   - 左上角子图像：0 1 1 0（结构度为 5，因为它进一步分割为四个子图像，每个子图像结构度为 1，总和为 4，加上自身分割，总结构度为 5）
   - 右上角子图像：1 1 1 1（结构度为 1，因为所有像素点颜色相同）
   - 左下角子图像：1 1 1 0（结构度为 3，因为它可以分割为两个子图像，一个全为 1，一个不全为 1）
   - 右下角子图像：1 0 1 1（结构度为 3，因为它可以分割为两个子图像，一个全为 1，一个不全为 1）

2. **计算总结构度**：5 + 1 + 3 + 3 = 12

（2）：

可以采用分治法的策略解决。分治法是一种将问题分解为更小的子问题来解决的策略，每个子问题都与原始问题相似，但规模更小。通过递归地解决这些子问题，然后将它们的解合并，可以得到原始问题的解。

（3）：

核心代码：

```
int calculateStructureDegree(const vector<vector<int>>& image, int x, int y, int size) {
    // 基本情况：如果子图像只有一个像素点，结构度为 1
    if (size == 1) {
        return 1;
    }

    // 检查当前子图像的所有像素是否相同
    int firstPixel = image[x][y];
    bool allSame = true;
    for (int i = x; i < x + size; ++i) {
        for (int j = y; j < y + size; ++j) {
            if (image[i][j] != firstPixel) {
                allSame = false;
                break;
            }
        }
        if (!allSame) {
            break;
        }
    }

    // 如果所有像素相同，结构度为 1
    if (allSame) {
        return 1;
    }

    // 如果像素不相同，将子图像分为四个更小的子图像，递归计算
    int subSize = size / 2;
    int structureDegree = calculateStructureDegree(image, x, y, subSize) +
    calculateStructureDegree(image, x, y + subSize, subSize) +
    calculateStructureDegree(image, x + subSize, y, subSize) +
    calculateStructureDegree(image, x + subSize, y + subSize, subSize);

    return structureDegree + 1;
}
```

1. **终止条件**:

   如果当前子图像的大小为 1x1（即只有一个像素点），则其结构度为 1。这是递归的终止条件。

2. **检查像素是否相同**:

   从当前子图像的左上角像素点开始，检查所有像素点是否与第一个像素点相同。

   如果发现任何一个像素点与第一个像素点不同，则 allSame 标志设为 false，并退出循环。

3. **处理所有像素相同的子图像**:

   如果当前子图像的所有像素点颜色相同，根据定义，其结构度为 1。

4. **分割子图像并递归计算**:

   如果当前子图像的像素点不完全相同，则将当前子图像分割为四个更小的子图像:

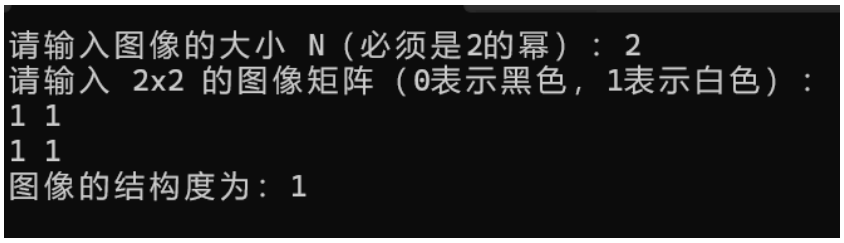
   - 左上子图像：左上角为 (x, y)，大小为 subSize x subSize。
   - 右上子图像：左上角为 (x, y + subSize)，大小为 subSize x subSize。
   - 左下子图像：左上角为 (x + subSize, y)，大小为 subSize x subSize。
   - 右下子图像：左上角为 (x + subSize, y + subSize)，大小为 subSize x subSize。

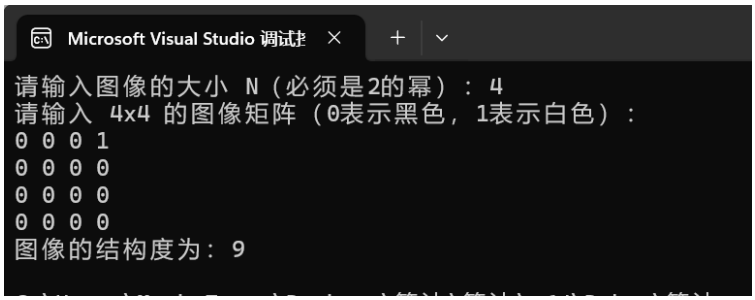
   对这四个更小的子图像分别递归调用 calculateStructureDegree 函数，计算它们的结构度。

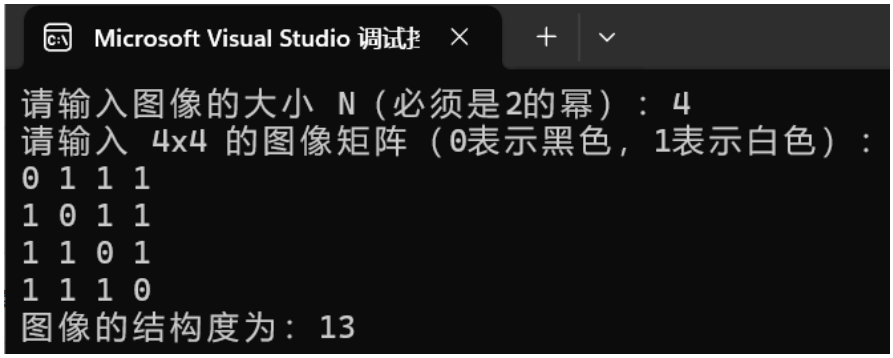
5. **计算当前子图像的结构度**:

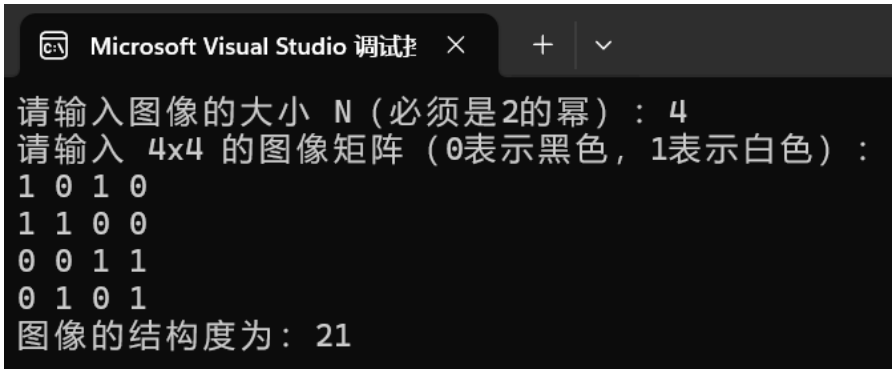
   根据定义，当前子图像的结构度等于四个更小子图像的结构度之和，再加上 1。

（4）









（5）

在最坏情况下，每个子图像都需要进一步分割，直到分割到单个像素点。对于大小为 $N \times N$ 的图像，递归深度为 $\log 2N$，每层递归需要检查 $N2$ 个像素点。因此，时间复杂度为 $O(N2\log N)$。