1.1
1、7 个位图
2、第 7 个位图
3、1024*2048*7/8 byte 约等于 1.8M
1.2
4-path 不存在
从 q 点出发无法找到与 q 点 4 邻接的点
N4(q) = {q} 不与 N4(q)有交集
所以无法找到 4-path
8-path 存在，其长度为 4
m-path 存在，其长度为 5
1.3
A 交 B 交 C
(A 交 B)并(B 交 C)并(A 交 C)
(B 交(~A)交(~C))并(A 交 C 交(～B))
2.1
利用双线性插值法求解，
由于实训用过 java 进行过图像处理，所以这次也同样用 java 进行图像处理。
用 ImageIO 和 BufferedImage 进行读取图片，并获取像素值存储进矩阵。
接着用 downScale(int w, int h, BufferedImage img) 进行图像缩放。
缩放利用了双线性插值方法。
双线性利用目标像素坐标，根据比例映射到原图像的浮点像素坐标，由于像素矩阵都是离散的，所以浮点像素坐标需找接近浮点像素最近的四个值，由于四个值可以映射到 3 维平面上，z 值相当于 RGB 值，四个点相连形成的平面中找到浮点坐标对应的值。
4 个最近的点分别可以映射成为(0,0), (0,1),(1,0),(1,1),浮点数坐标保留小数部分
先在 x 方向上找出插值
其四个坐标（0，f(0,1)），（0，f(0,0)），（1,f(1,0)），（1,f(1,1)），

Z1(x) = (f(1,0)-f(0,0))(x-0)+f(0,0)
Z2(x) = (f(1,1)-f(0,1))(x-0)+f(0,1)
其获得两个点分别为(Z1(x),0),(Z2(x),1)
再在 y 方向上找出插值
f(x,y) = (Z2(x)-Z1(x)) (y-0)+Z1(x)

根据该公式便能求出目标像素值。
下面为实现代码

f（x,y）=f(0,0)(1-x)(1-y)+f(1,0)x(1-y)+f(0,1)(1-x)y+f(1,1)xy

注意，这里按比例求出的浮点数后映射成的 4 个值可能超出高度和宽度的范围，要进行 min(width,y)及 min(height, x)才行，保证不越界。

```java
//缩放图片
public void downScale(int w, int h, BufferedImage bi) {
    BufferedImage buffImage =  new BufferedImage(w, h, BufferedImage.TYPE_INT_BGR);

    // 将图像画到buffImage中
    Graphics2D bGr = buffImage.createGraphics();
    bGr.drawImage(bi, 0, 0, w, h, null);
    bGr.dispose();

    //与原图相比，长宽比例
    double colRatio = (double)(width) / (double)(w);
    double rowRatio = (double)(height) / (double)(h);

    int[][][] downScaleRGB = new int[w][h][4];

    //双线性插值法
    for (int col = 0; col < w; col++) {
        //横坐标映射，并截取小数部分
        int k = (int)(col * colRatio);
        double u = (double)(col) * colRatio - k;
        for (int row = 0; row < h; row++) {
            //纵坐标映射，并截取小数部分
            int j = (int)(row * rowRatio);
            double c = (double)(row) * rowRatio - j;

            for (int i = 0; i < 4; i++) {
                //这里映射时的坐标，可能刚好超过最大高度及宽度，超过时像素点就设为最高的像素点及最宽的像素点
                int Q11 = (int)(sourceRGB[k][j][i]*(1.0 - c) + sourceRGB[k][j+1 >= height?height-1:j+1][i] *1.0* c);
                int Q22 = (int)(sourceRGB[k+1 >=width?width-1:k+1][j][i] * (1.0 - c) + sourceRGB[k+1 >=width?width-1:k+1][j+1 >= height?height-1:j+1][i] *1.0* c);

                // Dxy 为通过两次插值得到的目的像素坐标上的灰度值。
                downScaleRGB[col][row][i] = (int)(Q11 * (1.0 - u) + Q22 *1.0 * u);
            }
        }
    }

    //将RGB值重组成一个BYTE字节,并转化成一维数组
    int[] rgbArray = downScaleRGBToInt(w, h, downScaleRGB);
    // 把图像设置为新的RGB值
    buffImage.setRGB(0, 0, w, h, rgbArray, 0, w);
    // 将图片输出为png格式
    File output= new File("./src/92_" + Integer.toString(w) + "_" + Integer.toString(h) + ".png");
    try {
        ImageIO.write(buffImage, "png", output);
    } catch(Exception e) {
        return;
    }

}
```

12*8

24*16

48*32

96*64

192*128

300*200

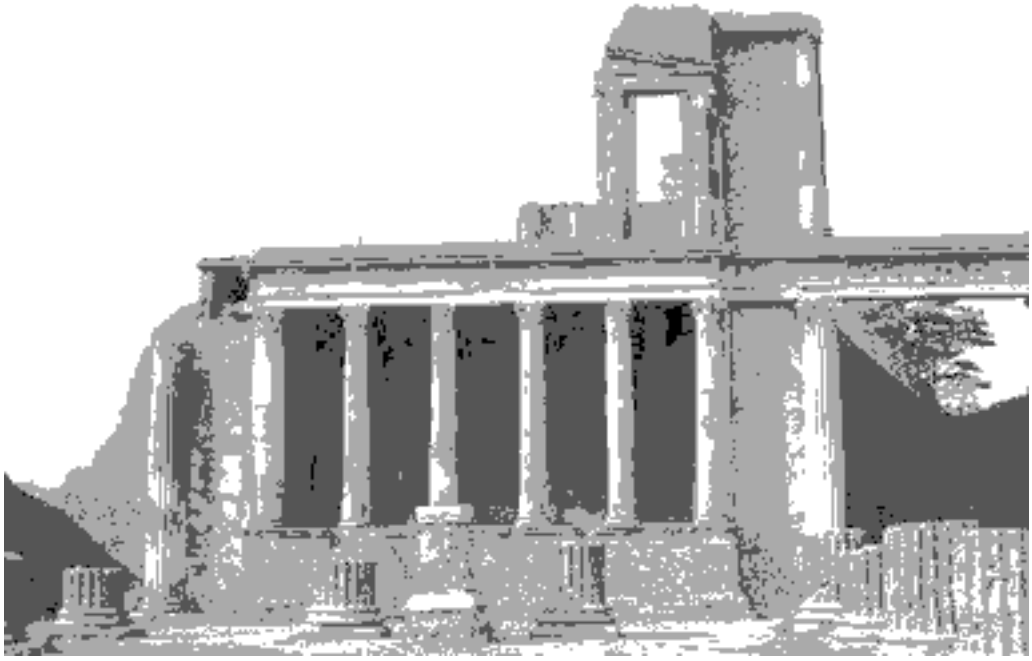
450*300

500*200

2.3
量化方法：
2level:保留像素值 0，255
4level: 0，85，170，255
8level:0, 36, 72, 108, 144, 180, 216, 252
分析可看出其间隔区间为 gap = 255/(level-1)
然后读取矩阵中的值，看其最接近的像素值，然后进行赋值，便得到满足要求的灰度图。
给定一个 RGB 值 temp，转化为对应灰度级别的值
思想如下：
先找出该值的下界
rgb = temp/gap*gap,
再根据 temp-rgb 与 gap/2 值相比，如大于等于就加 1，小于就不加。
注意：这里进行计算的时候可能会超过 255，所以求出来的值还要进行 min(rgb,255)求值，保证不越界。

2 level

4 level



8 level

32 level

128 level