HW1: Fundamentals

11331049 戴嘉华

1. Exercises

1.1
1) 256
2) The 256^{th} plane (which is the highest-order) is the most visually significant one.
3) 8 * 2048 * 2048 / 8 = 2^{22} bytes

1.2
4-path doesn't exist, for none of $N_4(P)$ is in V.
The shortest length of 8-path is 4.
The shortest length of m-path is 5.

1.3
$A \cap B \cap C$
$(A \cap B) \cup (A \cap C) \cap (B \cup C)$
$(A \cap C) \cup (B - (B \cap C) \cup (B \cap A))$

2. Programming

2.2

1.

 (192x128)

 (96x64)

 (48x32)

 (24x16)

 (12x8)

2.

 (300x200)

3.

 (450x300)

4.


(500x200)

5.

图片 Scale 算法有很多种，如 Bilinear 算法，Bicubic 算法，Nearest Neighbor 算法。我采用的是最简单直接的 Nearest Neighbor 算法。

我的思路是这样的：每个像素都有一个对应的坐标(x, y)，如果把一张图片 a 看做是另外一张图片 b 的拉伸变换的话。那么通过两张图片的宽度比例，图片 a 的每一个像素的 x 坐标都可以在图片 b 上找到对应的 x'：x' = (x / width_a) * with_b。y 坐标也同理：y'=(y / height_a) * height_b。

所以，我们就可以通过这种方式来找到目标变化图片和原图片上像素对应，然后把像素找到复制过去即可。核心代码：

```
output_img = Image.new(input_img.mode, size)
origin_width, origin_height = input_img.size
target_width, target_height = output_img.size
for i in xrange(0, target_height):
    for j in xrange(0, target_width):
        cp_i = int(float(i) / target_height * origin_height)
        cp_j = int(float(j) / target_width * origin_width)
        output_img.putpixel((j, i), input_img.getpixel((cp_j, cp_i)))
return output_img
```

2.3

1.

 (128 levels)

 (32 levels)

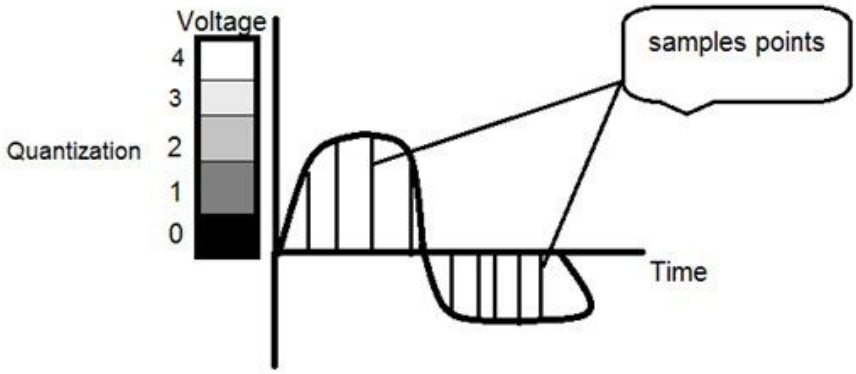
 (8 levels)

 (4 levels)

 (2 levels)

2.

算法思路其实比较简单：把[0, 255]分成不同的 level，每个 level 给一个颜色值。然后找出某个特定的颜色值在那个 level 中，就把这个 level 的颜色值赋给该颜色。



可以简单把原来[0, 255]想象成图上的连续曲线的 y 轴，quantization 就是把 y 轴分成不同层次，某个颜色值对应该层次的颜色值。就可以达到 quantize 的效果。

```
pi = input_image.getpixel((x, y))
gap = 256 / level # 一个层次有多少个像素
color_stop = 255.0 / (level - 1) # 一个层次的颜色跨度
target_pi = int(pi / gap * color_stop) # 算出该像素颜色对应的层次的颜色
output_image.putpixel((x, y), target_pi) # 覆盖原来的颜色
```