实验二 赫夫曼编解码

一、 实验目的

编写函数实现对输入图像进行赫夫曼编解码

二、 实验内容

a) 分别利用自定义函数和系统提供的函数对一幅图像进行赫夫曼编解码

b) 计算压缩率和平均码长。

c) 不能基于已有的算法库实现上述功能，但可参考开源代码。

三、 实验原理与理论基础

霍夫曼编码的原理为出现次数多的信源符号分配较短的码长，出现次数较少的信源符号分配较长的码长，从而降低平均码长达到压缩的目的。具体实现方式为统计所有信源符号的出现次数，初始将所有的信源符号作为一个叶子节点，每次构建新的节点时都选取当前无父结点的结点中频次最小的两个来构建，以此从下往上构建一棵霍夫曼树，当非叶子节点数目为叶子节点数目减一时即构建完成。由此霍夫曼树的每个节点都对应一个信源符号，假定从根节点往下行进时左右分别代表 0 或 1 即可由此路径求出霍夫曼码书，此方法保证了任何一个信源符号对应的霍夫曼编码都不是另一个的前缀。

相对于传统的霍夫曼编码，本次实验对于查找频次最小的无父结点部分进行了调整，将原先遍历每一个节点判断是否有父节点再比较大小调整为了增设一列表记录无父节点的索引，每当有节点的“父关系”发生变化时就进行调整，由此增加了空间复杂度但换来的时查找速度的加快。

将霍夫曼编解码作用于图像上就是对图像的灰度值作为信源符号集，统计频次，再将灰度值替换为对应的霍夫曼编码，此时得到的“图像”已成为 0、1

交替的序列，重新转换为 uint8 或者其他数据类型存储，即可在打乱原有像素排列的情况下减小数据量，如原来 3byte 空间只能存储 3 个位置的灰度值（特指 8bit 灰度图像），现在可以存储 3 个位置的灰度值和第四个位置灰度值的一部分。

四、 实验结果

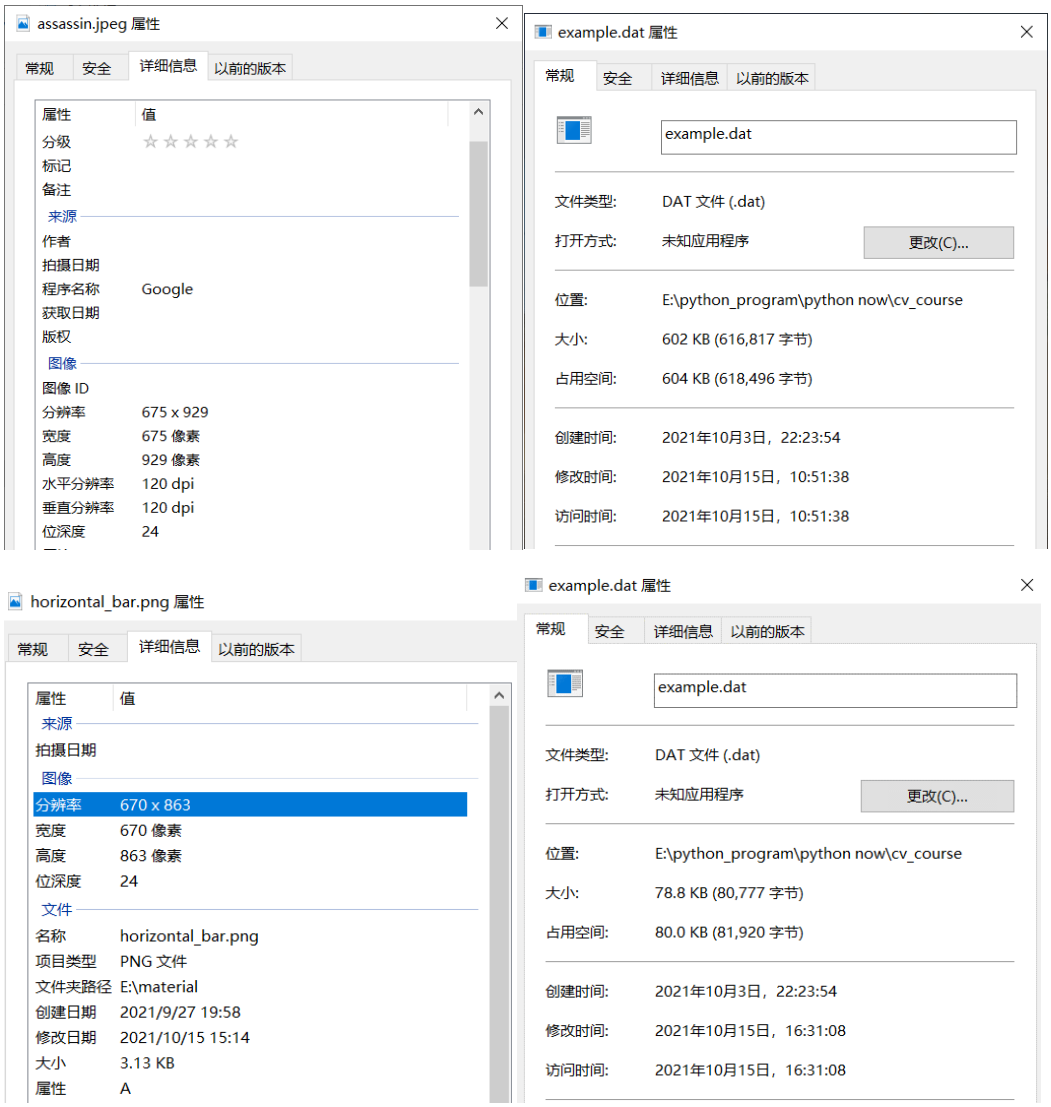

图 1. 原图像和其霍夫曼编码二进制文件对比

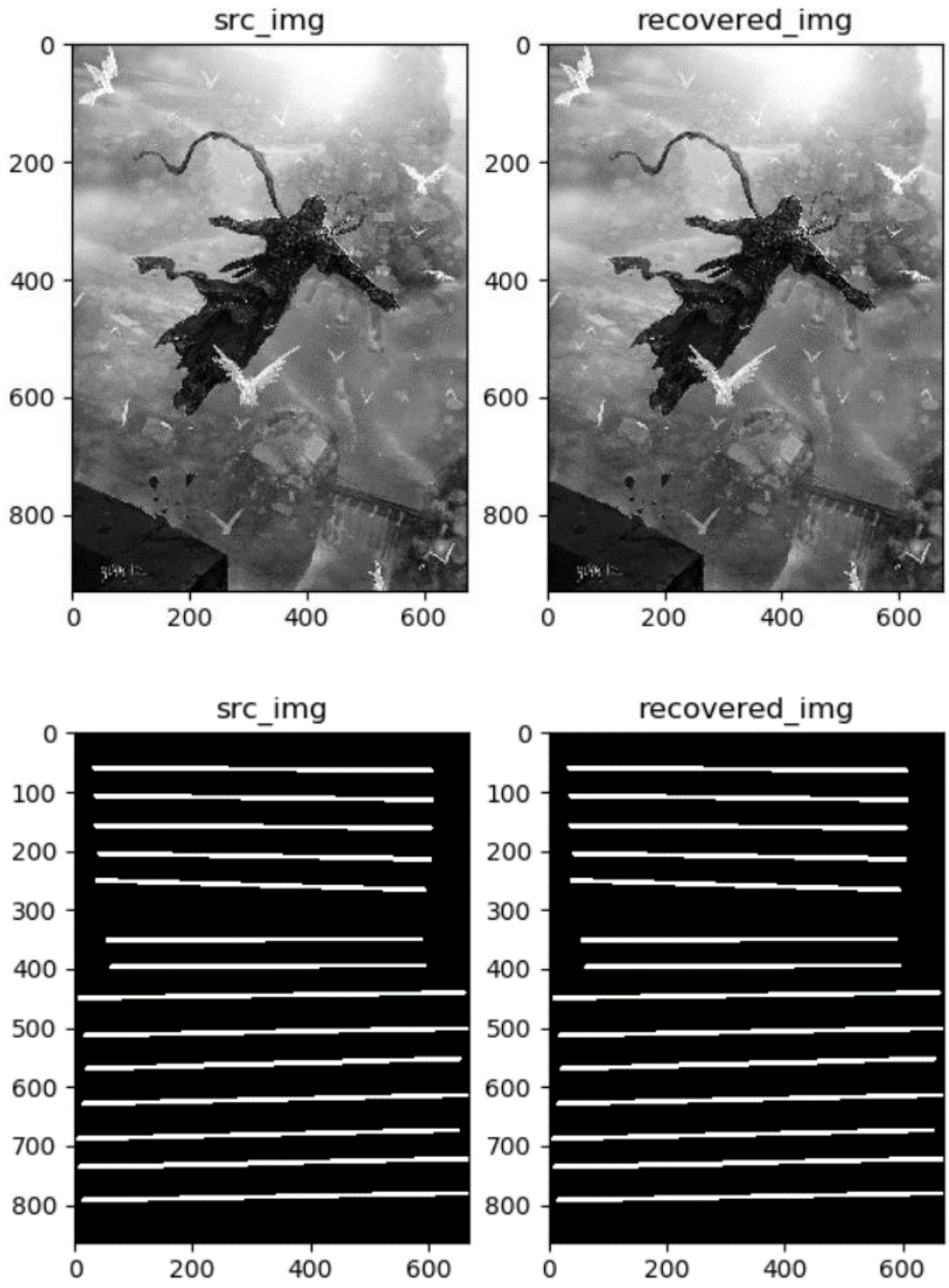

图 2. 原图和霍夫曼解码图像

五、 实验结果分析

实验使用了 2 张图片作为素材，一张富含多种灰度信息且分布较为平均（如图 2 左上），一张灰度信息贫乏且分布两极化（如图 2 左下）。分别按如下公式计算平均码长：

$$H(x) = \sum_{i=0}^{M} P(x_i) * len\big(h(x_i)\big)$$

其中 M 为出现灰度值数目减一，通常为 255；$h(x_i)$表明对该灰度值进行霍夫曼编码。计算得到两张图霍夫编码的平均码长为 7.87、1.12；理论压缩率分别为:1.016，7.134。这一结果是符合实际的，因为霍夫曼编码利用的就是信源符号出现频次的悬殊进行压缩，当信源符号分布较为均匀时压缩率较低

图 1 给出了两图像原始信息和编码后二进制文件的信息，以此可计算实际压缩率：

$$675 * 929 * 1/616817 = 1.016$$
$$670 * 863 * 1/80777 = 7.158$$

与理论值大致相等，证明我们的计算有效。

图 2 右上右下给出了由霍夫曼解码之后得到的图像，与原图无异，是无损压缩，同样通过计算编解码前后图像做绝对值差分图求和为 0，验证了我们的实验成功。

六、 附录

代码中主要函数有 huffman_coding(), huffman_tree(), huffman_decode(), encode(), write_bin(), read_bin()。前四个函数都是较为标准的霍夫曼编解码所需函数，具体思路也已在实验原理中解释过，不再赘述。后两个函数用于将图像的灰度值转换为对应霍夫曼编码后保存读取，write_bin()的实现方式为将图像的每一个灰度值都按顺序转为对应的 01 编码，此时一个灰度值对应的码长不再固定的 8 位，而是会随频次发生变化，而后将图像对应的一整个 01 码组重新 reshape 为 8 列的矩阵（如果码组大小不满足 8 的整数倍则在其后补零），重新将每 8 个 01 码读取为 uint8 大小的数保存，在末尾记录补零的个数。由此完成了图像的霍夫曼编码。read_bin()的过程为上述逆过程，每读取一个 uint8 类型

的数就转为对应 01 编码，删去最后一个数所记录的补零位数即完成了霍夫曼码的读取，送到 decoder 模块完成解码。