**北 京 林 业 大 学**

**2022 学年— 2023 学年第 二 学期 数字媒体技术基础 实验报告书**

实验地点： 学研T01 任课教师： 王晗

实验题目： 实验 4 图像压缩编码

实验环境： Windows、Python

实验目的：

1．熟悉 Python 程序的开发环境；

2．能够编辑、编译、运行简单的 Python 程序，掌握 Python 程序的基本结 构；

3．具备根据实际需要，设计选择合适的数据压缩编码方式的能力；

4．掌握数据压缩的基本理论，能够正确选择霍夫曼编码以及游程编码等熵 编码方法，以及根据压缩结果对编码效率进行分析。

实验内容：

1. 根据《PYTHON 实现图像的霍夫曼编码和译码》实验指导书内容对两张 不同的图片实现霍夫曼编码以及根据所设定的编码规则对压缩后的编码 数据进行解码还原图像。

2. 根据编码结果对编码效率进行比较和分析。

# 实现方法:

## 设计思想：

霍夫曼编码（Huffman Coding）是一种常用的无损数据压缩算法，可以对数据进行有效地压缩和解压缩。使用其进行图像压缩有以下优点：

1. 无损压缩：霍夫曼算法是一种无损数据压缩算法，它能够保证压缩后的数据可以完全恢复为原始数据，没有任何信息丢失。这在图像处理中是非常重要的，因为我们希望压缩后的图像在解压缩后能够与原始图像一致，没有任何质量损失。
2. 自适应编码：霍夫曼算法具有自适应性，即根据数据的频率分布动态调整编码方式。对于图像来说，不同像素值的出现频率是不同的，一些像素值可能更常见，而另一些像素值可能较少出现。通过使用霍夫曼编码，我们可以为常见的像素值分配较短的编码，而为不常见的像素值分配较长的编码，从而实现更高效的压缩。
3. 可变长度编码：霍夫曼编码是一种可变长度编码，它将不同的像素值映射为不同长度的编码。相比于固定长度编码（如等长编码），可变长度编码可以更好地利用编码空间。常见的像素值使用较短的编码，而不常见的像素值使用较长的编码，这样可以大幅度减小整体编码长度，实现更高的压缩比。
4. 压缩效率高：霍夫曼编码可以根据数据的统计特性进行编码，从而达到较高的压缩效率。对于图像来说，通常有很多相邻的像素值是相似的，霍夫曼编码可以利用这种相似性进行更有效的编码，减少存储空间的占用。

综上所述，使用霍夫曼算法来实现图像压缩具有无损性、自适应性、可变长度编码以及高压缩效率的优势。这使得霍夫曼算法成为图像压缩中常用的算法之一。

## 程序结构及流程简述

### 图像处理模块：

在window系统中安装PIL模块利用python的PIL模块处理图像。

### 构造节点的类：

先定义一个类，利用构造函数初始化其中的参数：

* 将其左结点，右结点，父节点设置为空
* 将结点的权重设置为0,
* 将其编码设置为空。

### 定义类内函数：将彩色图转为灰色图，（此时图像的每个像素点可以用单独的像素值表示）

1. 打开图片
2. 将bmp图片转换为灰值图
3. 保存灰值图像
4. 返回转换后的图片对象

### 定义类内函数：统计每个像素出现的次数：

1. 先定义一个字典，存储每个像素值出现的频率。
2. 遍历每个像素：

* 若此像素点不在字符频率字典里则直接添加
* 若存在在字符频率字典里则对应值加 1

1. 返回存储灰度值以及对应频率的字典

### 定义类内函数：构造叶子节点，分别赋予其值和对应的权值

1. 定义一个node\_list"的空列表。我们将使用这个列表来存储由像素值和权重创建的节点对象。
2. 该函数接受一个参数"pixel\_number"，它是一个元组列表，其中每个元组都包含两个值：一个整数表示像素值，一个浮点数表示相应像素值的权重。通过循环遍历所有元组，我们可以逐个处理每个节点，并生成一个节点列表。
3. 在循环体内，我们首先使用像素值作为节点代码的字符串表示（即每个节点的“名称”），并将其存储在变量"code"中。然后，我们将相应元组中的权重作为节点权重值存储在变量"weight"中。接着，我们使用这些值来创建一个新的节点对象，它们被添加到一个名为"node\_list"的列表中。
4. 最终，函数返回由所有新建立的节点组成的列表"node\_list"，这样我们就可以使用这些节点来进行后续的机器学习或深度学习任务。

### 定义类内函数：根据叶子结点列表，生成对应的霍夫曼编码树

1. 首先将待处理的节点按照权值从小到大排序
2. 当结点列表中的结点大于等于俩个的时候，执行如下的循环：

* 每次取最小权值的两个像素点进行合并
* 合并后的新节点权值为两个子节点权值之和
* 将两个子节点作为新节点的左右子节点
* 更新子节点的父节点为新节点
* 从节点列表中移除已经合并的子节点，并加入新生成的节点
* 将每次更新后的node列表按权值进行排序

1. 返回更新之后的节点列表

### 定义编码函数，返回编码表以及编码结果

1. 从参数中读取到已经载入的图片，并进行如下操作：
2. 得到图片的宽度和高度
3. 将像素点保存在新定义的list列表中，原来的二维矩阵变为一维数组
4. 调用先前定义的函数，统计每个像素点的次数，并根据出现的次数由小到大排序
5. 调用先前定义的函数，根据像素点的值和其出现次数构造节点list
6. 构造霍夫曼树,保存头结点
7. 构造编码表：创建一个空字典，用于存储每个像素值的编码
   * 1. 遍历霍夫曼树的叶子节点
     2. 为当前节点的像素值设置默认编码为空字符串
     3. 如果新节点不是头结点，则继续往上遍历:

* 如果新节点在父节点的左子树中, 则将编码表中该像素值的编码前面添加一个"1"
* 如果新节点在父节点的右子树中,则将编码表中该像素值的编码前面添加一个"0"
* 将当前节点赋值为其父节点，继续往上遍历

1. 输出每个像素点灰度值和编码
2. 输出编码表：
3. 创建一个空字符串，用于存储图像的编码结果
4. 遍历编码表，获取每个键值对：

如果像素值与当前键相等，将当前键对应的编码添加到编码结果字符串中

1. 将编码结果字符串写入文件中
2. 返回图片宽度、高度、编码表和编码结果字符串

### 还原原始的bmp图像，将编码结果解码成图像

1. 循环读取霍夫曼编码结果
2. 每次往后读一位：
   1. 遍历编码表：如果当前读到的编码在编码表里存在，将该编码对应的像素值添加到new\_pixel列表中，将当前读到的编码清空
   2. 若未找到则加上后续一个字符，继续查找
   3. 继续读取下一个编码
3. 然后使用两个嵌套的for循环遍历每一个像素位置，将存储新图像信息的new\_pixel列表中的像素值赋值给新图像的相应位置。
4. 最后使用save()方法将新图像保存为decode.bmp文件，并输出一条提示信息。

### 定义主函数：

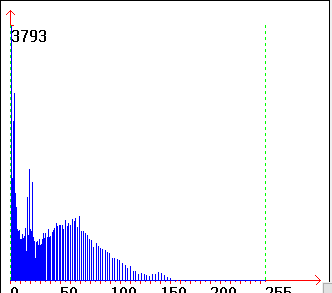
1. 首先调用了名为picture\_convert的函数，将位于'images/couple.bmp'路径下的一张图片转换成新的BMP格式，并保存在'images/new.bmp'路径下。
2. 接着，函数Huffman\_Coding会对新生成的图片进行哈夫曼编码（Huffman coding）。函数Huffman\_Coding返回四个变量：图片的宽度、高度、编码表和编码结果。
3. 最后，程序调用Decoding函数对编码结果进行解码，还原出原始的图像。

### 根据编码结果对编码效率进行比较和分析

哈夫曼编码是一种常用的数据压缩技术，通过根据字符出现的频率进行编码，使得出现频率高的字符使用较短的编码，出现频率低的字符使用较长的编码，从而实现对数据的高效压缩。下面我将对哈夫曼编码的编码效率进行比较和分析。

编码效率是指通过哈夫曼编码后所得到的编码长度与原始数据长度的比值。在进行哈夫曼编码时，出现频率高的字符使用较短的编码，而出现频率低的字符使用较长的编码，这种编码方式可以有效地减小编码后的数据长度，从而提高编码效率。

具体来说，如果原始数据中存在一些出现频率非常高的字符，那么通过哈夫曼编码可以将这些字符用较短的编码表示，从而大幅度减小编码后的数据长度。相反，如果原始数据中存在一些出现频率较低的字符，那么通过哈夫曼编码将这些字符用较长的编码表示，虽然会增加编码后的数据长度，但总体上仍然可以实现较高的压缩效率。



该图像的灰度直方图

可以看出来，这幅图的像素的灰度值较为集中，有一些出现频率非常高的字符，因此，运用哈夫曼算法编码可以大幅减小编码后的数据长度。，由此体现了哈夫曼编码的优越性。

利用Python编程，实现了对于哈夫曼编码的平均长度的计算，在统计中，共有114种不同灰度值的像素，若不进行哈夫曼编码，则需要的平均编码长度为7，哈夫曼编码缩短了编码的长度！。

需要注意的是，哈夫曼编码的编码效率与原始数据的特点密切相关。如果原始数据中存在大量的重复模式或者特定的规律，那么哈夫曼编码可能无法发挥出较高的压缩效果，因为在这种情况下，字符的出现频率可能比较均匀，使得编码后的数据长度与原始数据长度相差不大。

总结而言，哈夫曼编码在大多数情况下能够实现较高的编码效率，特别是对于存在频率较高的字符的数据。然而，在某些特殊情况下，哈夫曼编码可能无法达到理想的压缩效果。因此，在实际应用中，需要综合考虑数据的特点以及其他压缩算法的性能，选择合适的编码方式来进行数据压缩。

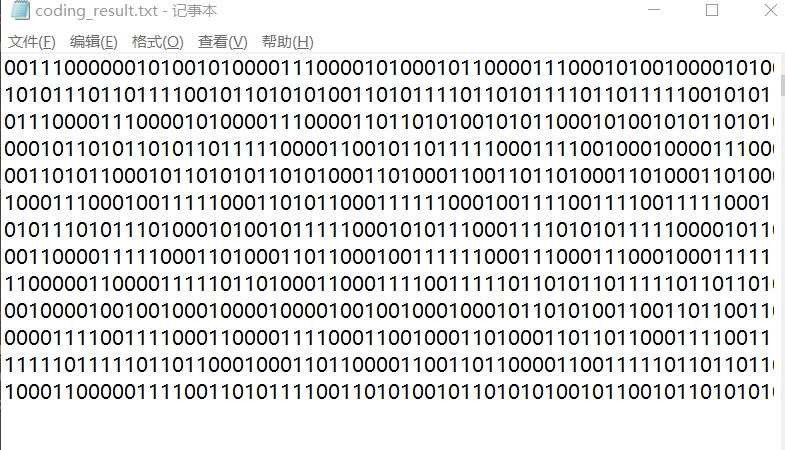
## 实验结果：

### 图像转化，生成new.bmp:



### 生成的编码表：





### 生成的新图像（decode.bmp）：



1. **实验中遇到的问题及相关思考**
2. 问题：

哈夫曼编码在之前数据结构的课上进行过深入的学习，因此没有什么问题

2-相关思考：

哈夫曼编码是一种无损压缩的编码方式，是一种最佳变长码，其平均码长接近于熵值。

但是，这幅图片中没有体会，但是当图片数据成分复杂的时候，庞大的信息源会让哈夫曼编码的编码表比较大，码表生成的计算量增加，降低了编译码的速度。

1. **实验心得体会**

通过这次实验，让我对哈夫曼编码以及其应用有了更加深刻的理解，哈夫曼编码的效率于我而言不再是简单的编码平均长度的减少，而是可以体会到的图片占用存储空间的减少。通过分析老师的代码加深了对其的理解，在未来做大创的时候能够运用的更加得心应手。