**深 圳 大 学 实 验 报 告**

**课程名称：­ 多媒体系统导论**

**实验项目名称： 图像压缩编码实验**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术（创新班）**

**指导教师： 朱映映**

**报告人： 何泽锋 学号： 2022150221**

**实验时间： 2025年5月15日-2025年5月19日**

**实验报告提交时间： 2025年5月18日**

**教务部制**

|  |  |  |
| --- | --- | --- |
| **一、实验目的：**  1. 掌握无损图像数据的读写方法；  2. 掌握图像无损压缩编码压缩技术的基本原理；  3. 编码实现霍夫曼编码，算术编码，词典编码（编程语言不限）；  4. 编程计算图像的熵、编码后的实际压缩比。 | | |
| **二、实验环境：**  系统：Windows 11  编程语言：python  依赖库：   |  | | --- | | import os  import time  import math  import struct  from collections import Counter  from fractions import Fraction  from PIL import Image  from heapq import heappush, heappop | | | |
| **三、实验过程及内容：**  **1. 霍夫曼编码**    图1 霍夫曼编码过程 图2 霍夫曼解码过程  （1）编码原理：   |  | | --- | | 霍夫曼编码通过构建霍夫曼树实现变长编码，核心步骤为：  ①频率统计：统计图像各通道像素值的出现频率。  ②优先队列构建：将频率作为权重，构建最小优先队列。  ③树构建：每次取出两个最小权重节点合并为父节点，直至队列只剩一个根节点。  ④编码生成：从根节点到叶子节点的路径，左子树为0，右子树为1，生成唯一前缀编码 |   （2）代码实现：   |  | | --- | | ①编码部分  首先构建霍夫曼树结构，用于存储节点信息    图3 霍夫曼树节点结构  统计像素频率    图4 像素频率统计代码  然后使用优先队列构造霍夫曼树，流程如下：  ·循环条件：持续合并直至堆中仅剩一个根节点。  ·合并步骤：  a. 取出频率最小的两个节点  b. 创建父节点，频率为两子节点之和  c. 将父节点重新插入堆中  当只有一个像素的时候特殊处理，直接构建根节点即可。    图5 优先队列构建霍夫曼树流程  ②生成编码表  通过递归遍历霍夫曼树生成每个像素值对应的二进制编码  ·首次调用时创建空字典code\_map存储编码结果。  ·递归遍历：  a. 左子树路径：添加0  b. 右子树路径：添加1  ·叶子节点处理：  a. 若节点包含像素值，将当前路径编码存入code\_map  b. 特殊情况：若整棵树只有一个节点，编码为0  ·返回结果：遍历完所有叶子节点后，返回完整的编码表。    图6 编码表生成代码  ③解码数据读入  读取头部信息  ·频率字典：用于重建霍夫曼树。  ·总位数：压缩后的二进制数据总长度（可能不足整数个字节）。  ·像素数：原始图像的像素总数，用于控制解码终止条件。    图7 头部信息读取代码  ④解码流程  重建霍夫曼树，传入频率表    图8 霍夫曼树重建代码  逐位解码：  ·按位遍历：从压缩数据中读取每一位，根据位值（0/1）决定向左 / 右子树移动。  ·叶子节点处理：  a. 若到达叶子节点（symbol 不为 None），记录该像素值。  b. 重置指针到根节点，继续解码下一个像素。  ·终止条件：  a. 读取完所有压缩位或已还原的像素数达到预期值。    图9 逐位解码流程 |   **2. LZW编码**    图10 LZW编码原理示意图 图11 LZW解码原理示意图  （1）编码原理   |  | | --- | | ①初始字典包含所有单个符号（此处是 0–255 的灰度值）；  ②逐步读取最长的已编码字符串 w，如果 w+c 在字典中，则继续延伸，否则：  ③输出 w 对应的字典索引；  ④将 w+c 加入字典（新索引）；  ⑤重置 w = c。 |   （2）代码实现   |  | | --- | | ①编码部分  初始化字典，将0-255的像素都放到字典中    图12 字典初始化代码  核心部分  ·扩展当前字符串：将当前像素追加到current\_str形成new\_str。  ·字典查找：  a. 若new\_str存在于字典中，更新current\_str继续累积。  b. 若不存在，则输出current\_str的编码，将new\_str添加到字典（若字典未满），并重置current\_str为当前像素。    图13 字符串扩展与字典查找代码  ②解码部分  读取头部信息，编码位数是每个编码占用的位数，决定字典最大容量    图14 头部信息读取代码  初始化指点你，与编码器相同，包含 256 个像素值    图15 字典重建代码  核心解码逻辑：  ·处理当前编码：  a. 若编码存在于字典中，直接获取对应字节串。  b. 若不存在（特殊情况），使用prev\_str + prev\_str[0]（见示例）。  ·更新字典：  添加新条目prev\_str + entry[0]  ·边界检查：  当解码像素数达到预期值时提前终止，防止越界。    图16 解码逻辑代码  ③编码解码差异  ·执行顺序：  编码：输出代码 → 添加新条目 → 处理下一个字符  解码：添加新条目 → 处理当前代码 → 更新前一个字符串 |   **3. 算术编码**    图17 算术编码流程图 图18 算术解码流程图  （1）编码原理   |  | | --- | | 算术编码将整个消息映射为 [0, 1) 区间内的一个小数，通过迭代缩小区间实现码：  ①概率区间划分：根据符号概率将当前区间划分为子区间，符号概率越大，子区间越长。  ②区间更新：每次读取符号后，将区间更新为对应子区间，直至处理完所有符号。  ③二进制转换：将最终区间转换为最短二进制字符串。 |   （2）代码实现   |  | | --- | | ①编码部分  ·对像素出现的次数进行统计，得到频率    图19 像素频率统计代码  ·累积频率：cum\_freq[i] 表示像素值 < i 的总出现次数，用于后续确定每个像素在当前区间内的子区间。    图20 累积频率计算代码  ·核心编码  ·对每个像素，根据其累积频率将当前区间 [low, high) 划分为子区间：  新下界：old\_low + 区间宽度 × cum[s]/total  新上界：old\_low + 区间宽度 × cum[s+1]/total    图21 区间划分与更新代码  计算最终的编码，取最终区间的中点 (low + high)/2作为代表值    图22 最终编码计算代码  ②解码部分  读取头部信息，并重建累计频率表    图23 累积频率表重建代码  解析编码结果    图24 解析编码结果  核心解码部分  ·计算相对位置：ratio表示tag在当前区间[old\_low, old\_high)中的相对位置  ·转换为累积频率：target = ratio × total，将相对位置映射到累积频率范围  ·符号查找：遍历累积频率表，找到target所在的区间[cum[s], cum[s+1])，确定对应符号 s。  ·区间更新：根据符号的累积频率缩小当前区间，与编码器一致。    图25 解码核心逻辑代码 |   4. 熵计算  （1）原理解释   |  | | --- | | 熵值越低，说明数据冗余度越高，可压缩空间越大。在图片压缩问题中，每个像素值可视为一个随机变量，可以根据如下公式计算出图像的熵值 |   （2）代码实现   |  | | --- | | 实验过程是对彩色图像进行处理，因此对每个通道分别处理。    图26 分通道处理  ·频率统计：使用 Counter 统计每个像素值的出现次数  ·概率计算：每个像素值的概率  ·逐项累加：对每个概率项计算 并求和    图27 熵计算代码 |   （3）结果展示   |  | | --- | | 实验采用的图像如下所示，大小为60×60的bmp图像，图像是从AIGC生成的图片中截取得到。    图28 原始图像  调用代码计算图像的熵，得到如下结果，简单分析可以知道，R通道的熵值最低，说明像素值分布相对集中，人物主体是红色衣服，主要集中在图像的中间区域。B通道熵值最高，在这张图里可能意味着蓝色相关元素的分布更复杂、变化多    图29 原始图像各通道信息熵 |   5. 压缩比计算  （1）原理解释   |  | | --- | | 通过比较压缩前后的比特数，评估压缩结果，计算公式如下所示：  本次实验处理的是较小的图像，直接比较压缩文件和原始文件可能会发现压缩比小于1，这是因为压缩算法生成的.bin文件包含：压缩后的数据、概率表/字典（霍夫曼编码表、算术编码的概率分布、LZW 的字典）、其他元数据（如图像尺寸、通道信息）。 |   （2）代码实现   |  | | --- | | ①原始图像计算逻辑：  ·假设每个像素用 8 位表示（标准 BMP 格式）  ·对于 RGB 图像，通道数为 3；灰度图像通道数为 1  ·原始数据总位数 = 像素数 × 通道数 × 8 位    图30 原始图像大小计算  ②编码结果大小计算  直接累加三个通道的字节数量    图31 压缩图像大小计算 |   （3）结果展示   |  | | --- | | ①霍夫曼编码    图32 霍夫曼编码压缩统计  ②算术编码    图33 算术编码压缩统计  ③LZW 编码    图34 LZW编码压缩统计  **压缩比**：算术编码 > 霍夫曼编码 > LZW编码  ·霍夫曼编码：压缩后数据大小为7133 B，压缩比为1.51:1。霍夫曼编码通过构建基于符号频率的最优二叉树来分配编码长度，对于像素值频率分布差异明显的图像，能有效利用频率特性，将高频像素值用短编码表示，低频用长编码，从而实现较好的压缩效果。在这张图像上，该算法表现出了相对不错的压缩能力。  ·算术编码：压缩后数据为7087 B ，压缩比达到1.52:1。算术编码不是对单个符号编码，而是将整个消息表示为[0, 1)区间内的一个小数，根据符号概率动态缩小区间。它能更精细地利用概率信息，理论上比霍夫曼编码压缩效率稍高，在这组数据中也得到了体现，其压缩后数据量略小于霍夫曼编码，压缩比稍高。  ·LZW编码：压缩后数据为12022 B ，压缩比是0.9:1。LZW编码基于字典，将首次出现的字符串映射为代码，随着字典动态扩展来压缩数据。在这张图像上，其压缩效果相对霍夫曼和算术编码较差，可能是图像内容的重复模式没有达到LZW编码能充分发挥优势的程度，导致字典扩展效率不高，未能有效降低数据量。  **运行时间**： 霍夫曼 < LZW编码 < 算术编码  ·霍夫曼编码：压缩耗时为0.0516秒，霍夫曼编码的计算主要集中在构建霍夫曼树和遍历树生成编码，算法逻辑相对清晰，计算量适中。  ·算术编码：压缩耗时为0.0826秒，明显高于霍夫曼和 LZW 编码。这是因为算术编码在计算过程中需要进行高精度的小数运算，对区间进行不断细分和更新，计算复杂度较高，尤其在处理大量数据时，计算量会显著增加，导致耗时较长。  ·LZW编码：压缩耗时为0.0575秒，在构建字典和编码过程中，虽然涉及字典的动态更新，但对于这张图像的数据量和特性，在当前实现下与霍夫曼编码耗时相当，说明其实现效率尚可，没有产生过多额外计算开销。 | | | |
| **四、实验结果与分析：**  表1 不同编码算法的压缩比较   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 编码算法 | 原始图像大小 | 压缩后数据大小 | 压缩比 | 压缩耗时 | | 霍夫曼编码 | 10800KB | 7133KB | 1.51 | 0.0516 | | 算术编码 | 10800KB | 7087KB | 1.52 | 0.0826 | | LZW 编码 | 10800KB | 12022KB | 0.9 | 0.0575 |   1. 霍夫曼编码  （1）实验结果：  ·原始图像大小为10800B，压缩后文件大小为7133B，压缩比为1.51  （2）分析：  ·霍夫曼编码通过变长编码减少了高频像素的存储空间，但低频像素的编码较长，可能导致压缩效率受限。  ·压缩比受图像内容影响较大，若图像中像素值分布均匀，压缩效果可能不佳。  ·速度优势：仅需一次频率统计和树构建，逻辑简单  2. LZW 编码  （1）实验结果：  ·原始图像大小为10800B，压缩后文件大小为12022 B，压缩比为0.9  （2）分析：  ·LZW 编码对重复序列的压缩效果显著，但图像中的连续性较差不能很好的发挥。  3. 算术编码  （1）实验结果：  · 原始图像大小为10800B，压缩后文件大小为7087 B，压缩比1.52，略高于霍夫曼，验证了 “理论压缩效率更优” 的特性。  （2）分析  **·**无码字冗余：无需为每个符号单独分配固定长度码字，直接输出小数的二进制表示，理论上可达熵极限  （3）问题：  ·耗时：每次更新区间需进行高精度浮点运算，需要很长的运行时间  ·高精度问题：需要保存高精度的浮点数，面临精度丢失问题，实验时进行多次修改才实现了高精度的存储，在面临大图片的时候处理较为麻烦。 |

|  |
| --- |
| **五、心得体会：**  1. 算法结合应用  ·霍夫曼编码：适合像素值分布集中、重复率中等的图像，优势在于实现简单、解码速度快（仅需遍历二叉树），但需注意频率表存储优化。  ·算术编码：适合概率分布明确、需极致压缩的场景，压缩效率理论最优，但计算复杂度高，需根据硬件性能选择浮点或整数实现方案。  ·LZW 编码：适合长重复序列突出的数据，但对自然图像压缩效果差，需结合数据特性提前优化字典策略。  2. 实验问题  ·LZW负压缩：算法选择必须基于图像内容分析，在一些问题上可能不能很好起作用。  ·浮点运算的精度问题和耗时问题表明，理论最优算法在实际实现中需要综合考虑，理论值未必是实际应用的最优解。 |
| **思考题：**  选择一个图像文件，针对该图像文件，讨论在上述三个无损编码中，哪一个无损压缩比最高，哪一个最低，并分析原因。  为了解决实验室出现的LZW压缩比小于1的问题，设计了如下图片进行测试  （1）选择图像  设计了如图所示非常简单与连续的三色图    图35 原始图像  （2）分析  在上述三个编码方式中，猜测LZW表现会最好，这是因为这张图的规律性很强，具有大量的连续性片段，并且颜色单一，不需要往字典中加入大量条目，码长很短。又根据资料算术编码基本都是优于霍夫曼编码，因此猜测霍夫曼的压缩比是最低的，因为霍夫曼没办法使用图像上的连续信息，本质还是对像素值在处理，而算术虽然也如此但是用大量的计算时间进一步压缩空间。  （3）实际结果  表2 特定图像下的压缩比较   |  |  |  | | --- | --- | --- | | 算法 | 压缩比 | 时间 | | 霍夫曼编码 | 4.90 | 0.0152 | | 算术编码 | 5.06 | 0.0243 | | LZW编码 | 13.93 | 0.0110 | |

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  **指导教师签字：**  **年 月 日** |
| **备注：** |

**注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。**

**2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内**。