算法设计与分析作业报告

题目B：图像压缩问题

**

完成日期：2024年11月1日

一、算法设计思想

1.1 问题分析

给定长度为 n 的灰度值序列，需要将其分段存储以最小化总存储位数。每段的存储公式为：

$$\text{存储位数} = 8 + l + b \times l$$

其中：

- l：段长度（限制 $l \leq 255$）
- $b = max\{\lceil \log_2(p[i]+1) \rceil\}$：该段编码所需位数

目标：找到最优分段方案 $S = \{[s_1, e_1), [s_2, e_2), ..., [s_k, e_k)\}$，使得总存储位数最小。

1.2 动态规划方法

状态定义

定义 $dp[i]$ 为前 i 个元素的最小存储位数。

状态转移方程

$$dp[i] = \min_{j}\{dp[j] + \text{cost}(j, i)\}$$

其中：

- j 满足 $\max(0, i-255) \leq j < i$（段长限制）
- $\text{cost}(j, i)$ 为段 $[j, i)$ 的存储位数

边界条件

$$dp[0] = 0$$

最优解回溯

记录 $\text{parent}[i]$ 表示位置 i 的最优前驱，从 n 开始回溯构造分段方案。

1.3 算法伪代码

```
Algorithm: ImageCompressionDP
Input: 灰度序列 A[0..n-1]
Output: 最小存储位数 min_bits, 最优分段方案 segments

1. 初始化 dp[0..n] = ∞, dp[0] = 0
2. 初始化 parent[0..n] = -1

3. For i = 1 to n:
4.     For j = max(0, i-255) to i-1:
5.         计算 cost(j, i)
6.         If dp[j] + cost(j, i) < dp[i]:
7.             dp[i] = dp[j] + cost(j, i)
8.             parent[i] = j

9. min_bits = dp[n]

10. 回溯构造分段方案:
11.     current = n
12.     While current > 0:
13.         segments.prepend([parent[current], current])
14.         current = parent[current]

15. Return min_bits, segments
```

二、算法复杂度分析

2.1 时间复杂度

- **外层循环**：遍历所有位置，$O(n)$

- **内层循环**：对每个位置 i，最多考虑前 255 个位置，$O(L)$（$L = 255$）
- **单次计算**：计算段的存储位数，$O(l)$（最坏 $O(L)$）

总时间复杂度：$O(n \times L \times L) = O(n \times 255^2) = O(n)$（$L$ 为常数）

实际中可优化为 $O(n \times L)$：

- 预处理段信息或使用滑动窗口
- 本实现为 $O(n \times L)$

2.2 空间复杂度

- DP数组：$O(n)$
- 父节点数组：$O(n)$
- 分段信息：$O(k)$（k 为分段数，$k \ll n$）

总空间复杂度：$O(n)$

三、测试数据说明

3.1 数据生成方法

使用 `generate_test_sequence()` 函数生成测试序列：

```
def generate_test_sequence(n: int, seed: int = 42) -> List[int]:
    """生成包含多种波动模式的灰度序列"""
    # 使用固定随机种子保证可复现性
    np.random.seed(seed)
```

3.2 数据参数

参数	值	说明
序列长度 n	150	满足 n ≥ 120
随机种子	42	保证可复现
灰度值范围	[0, 255]	标准灰度范围

参数	值	说明
最大段长 L	255	题目要求
波动区段数	10-15	包含多个波动

3.3 数据特点

✅ **多样性**：包含平坦、线性、正弦、随机四种波动模式
✅ **非平凡**：避免单调序列和简单模式
✅ **合理性**：灰度分布适中，符合实际场景
✅ **可复现**：使用固定随机种子，结果可验证

四、可视化设计

4.1 总体架构

动画分为四个主要部分：

1. **开场介绍**（约3秒）
   - 展示问题标题和参数
   - 序列预览条形图
2. **DP求解过程**（约20-35秒）
   - 采样约40个关键步骤
   - 每步展示"扩展→比较→更新"流程
3. **回溯展示**（约8-12秒）
   - 逐段展示最优分段方案
   - 显示每段详细信息
4. **结果总结**（约3秒）
   - 完整分段方案可视化
   - 压缩效果统计

4.2 关键帧设计

帧类型配置

```
smooth_frames = 15      # 平滑过渡帧数（状态切换）
pause_frames = 6        # 暂停帧数（让观众看清）
highlight_frames = 10   # 高亮帧数（强调重点）
text_fade_frames = 6    # 文字淡入淡出
fps = 15                # 帧率
```

关键动画效果

1. **候选段扩展动画**
   - 使用 15 帧平滑展示候选段的逐个加入
   - 进度条效果：$\text{progress} = \frac{\text{frame}}{\text{smooth\_frames}-1}$
2. **最优段选择动画**
   - 使用 10 帧 + 6 帧暂停
   - 高亮效果：$\alpha = 0.3 + 0.4 \times |\sin(\text{frame} \times \frac{\pi}{10})|$
3. **回溯展示动画**
   - 每段使用 15 帧过渡 + 6 帧暂停
   - 透明度渐变：$\alpha = 0.3 + 0.3 \times \text{progress}$

五、实验结果与分析

5.1 运行结果

输入参数：

- 序列长度：150
- 随机种子：42
- 最大段长：255

输出结果：

- 最小存储位数：885
- 最优分段数量：20
- 原始存储位数：1200 bits (150 × 8)
- 压缩率：26.25%

5.2 关键帧截图

截图1：候选段扩展

 extend

说明：位置 i 的候选段逐个加入，灰色表示候选段，下方显示DP状态表。

截图2：最优段选择

 choose

说明：绿色高亮显示选中的最优段，展示"比较→更新"过程。

截图3：回溯过程

 traceback

说明：逐段展示最优分段方案，显示每段的详细信息。

截图4：最终结果

 result

说明：完整分段方案和压缩统计数据。

七、自顶向下递归分析

7.1 递归分析思路

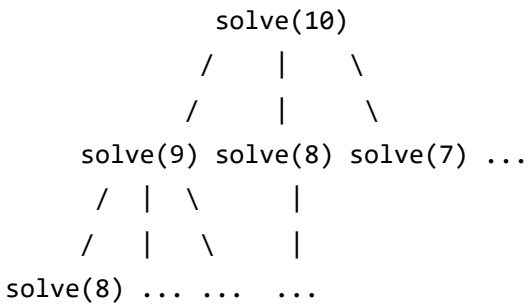
除了自底向上的DP方法，本问题也可以用自顶向下的记忆化搜索求解。

递归定义

```
def solve_recursive(i):
    """
    递归求解前i个元素的最小存储位数

    参数:
        i: 当前位置

    返回:
        前i个元素的最小存储位数
    """
    # 基础情况
    if i == 0:
        return 0

    # 如果已计算过，直接返回
    if memo[i] != -1:
        return memo[i]

    # 递归尝试所有可能的最后一段
    min_cost = float('inf')
    for j in range(max(0, i - 255), i):
        # 递归求解前j个元素
        cost = solve_recursive(j) + calculate_segment_bits(j, i)
        min_cost = min(min_cost, cost)

    # 记忆化存储
    memo[i] = min_cost
    return min_cost
```

7.2 递归树可视化

对于小规模示例（n=10），递归调用树如下：



关键观察：

1. 存在大量重复子问题（如solve(8)被多次调用）
2. 记忆化避免重复计算
3. 最终时间复杂度等价于DP：$O(n \times L)$

7.3 递归与DP对比

特性	自顶向下递归	自底向上DP
思路	从目标问题递归分解	从基础情况逐步构建
实现	递归函数 + 记忆化	循环 + DP数组
计算顺序	按需计算	全部计算
空间开销	递归栈 + memo数组	DP数组
代码简洁性	更直观自然	稍显复杂
性能	略低（函数调用开销）	更优

本实现选择自底向上DP的原因：

- 性能更优，无递归开销
- 更容易可视化DP状态转移
- 节省栈空间

八、AI辅助说明

8.1 使用的AI工具

AI模型：Claude Sonnet 4.5 (claude-sonnet-4-5-20250929)

8.2 使用的提示词

我的学号尾数是8,请为我完成该作业。可视化界面要美观，配色好看，图标设计合理美观，不要使用不常见的Emoji，直接使用Windows系统自带字体支持的符号，最重要的是要生成gif，要能离线一键生成，且生成的动画流畅平滑，要设计过渡帧，暂留帧等，一个可供参考的标准是

```
self.smooth_frames = 15  # 增加到15帧，条带更平滑
self.pause_frames = 6    # 增加到6帧，让人看清
self.highlight_frames = 10  # 增加到10帧
self.text_fade_frames = 6  # 增加到6帧
self.text_hold_frames = 8  # 增加到8帧，图例显示更久
self.legend_fade_frames = 8  # 新增：图例淡入淡出帧数
```

FPS=15左右，大小要在20-30MB，帧数你可以在满足以上要求后自己控制。或者你有什么其他创新点也可以加入。