数据建模第四次作业——碎纸片的拼接复原

学号：032201218 姓名：陈彦哲

目录

一、问题重述

随着计算机视觉与图像处理技术的不断发展，碎纸片自动拼接技术在司法鉴定、文献修复、军事侦察等领域中展现出重要应用价值。传统的人工拼接方法虽然准确率较高，但在碎片数量较多、图像结构复杂的情况下，人工处理效率低、主观性强，难以满足快速复原的需求。因此，如何借助计算机算法实现碎纸片的高效自动化拼接，成为当前研究的重要课题。

本研究拟针对不同碎纸类型与结构设计拼接模型与算法，并完成以下任务：

针对仅纵向切割的碎纸情形，建立适用于同一页单面印刷文字文档碎片的拼接复原模型与算法。要求对附件 1 与附件 2 中的中文、英文单页碎纸数据进行自动拼接；若拼接过程中存在难以判定的情况，需设计人工干预机制，明确其触发条件与插入时机，并以图像+表格形式展示最终拼接结果。

针对同时进行纵横切割的碎纸情形，构建二维网格式的图像拼接模型，能够处理纸张被切割成规则矩形块的情况。要求完成附件 3（中文）、附件 4（英文）文档的自动拼接，并结合人工辅助策略对难以自动判断的步骤进行干预，最终结果同样以图表形式输出。

进一步考虑双面打印文档的碎纸拼接问题，设计适用于正反面图像配对与联合拼接的算法模型。附件 5 提供了一页英文双面打印文档的碎纸数据，需对其进行自动配对、分类与拼接，分析正反面对复原过程带来的挑战，并输出最终图像与拼接编号表。

二、问题分析

随破碎文件的拼接在司法物证复原、历史文献修复、军事情报重建等领域具有重要意义。传统方法主要依靠人工完成，虽然准确率较高，但效率极低，尤其在碎片数量庞大、图像质量复杂的情形下，人工拼接难以在有限时间内完成任务。随着计算机视觉、图像处理等技术的发展，基于图像特征的自动拼接方法逐渐成为研究热点。

本题的核心在于设计适应不同碎片切割方式（仅纵切、横纵切、双面横纵切）下的自动拼接模型与算法，以实现碎纸片文档的快速重构。通过分析题目背景与附件中提供的图像数据，本题可抽象为以下两个关键任务：

1. **图像特征提取与匹配评估：**由于碎纸片在裁剪过程中边缘结构常遭破坏，传统基于轮廓、角点的形状匹配难以奏效，因此需转向图像灰度层面，提取左右边缘像素灰度信息，并结合边缘黑白过渡方向（上下边缘类型）与空白距离等特征，构建拼接相似度评估机制。
2. **拼接排序的组合优化问题：**问题一为仅纵切情形，可视为一维拼接路径寻找；问题二与三则涉及横纵裁剪与二维拼接排序，需进行图像聚类（分行）后分别在行内和行间进行拼接排序。在设计上可借鉴旅行商问题（TSP）的求解思想，或采用贪心策略近似搜索最优路径。

基于上述分析得出，本题属于典型的“图像处理 + 模式识别 + 组合优化”类建模问题，因此我们拟采用如下策略：

问题一：

基于左右边缘像素灰度差构建拼接误差评分函数，通过贪心算法获得最优拼接序列

问题二：引入上下边缘类型与距离特征进行图像分行聚类，行内拼接同问题一，行间排序基于上下边缘匹配度完成。

问题三：在二维拼接的基础上，引入正反面碎片配对机制，通过结构特征筛选初步配对对，再由人工确认判断进行最终修正，提升匹配准确性。

三、模型假设

为确保模型构建的合理性与可实现性，同时简化部分实际中难以建模的复杂因素，我们在分析题目背景与数据附件的基础上，提出如下模型假设：

1. **碎片来源一致性假设：**假设每个附件中的所有碎纸片均来自同一页纸，且该页纸被完整切割，碎片间不存在缺失或混入其他文件碎片的情况。
2. **碎片形状规范假设：**假设所有碎纸片均为大小一致的矩形图像，且边缘裁切规整，无旋转、扭曲、破裂、边缘撕裂等情况出现。
3. **图像方向统一假设：**假设所有碎纸片的图像方向一致，即均为“正放”，无需对碎纸片进行旋转校正处理。
4. **碎片编号有序假设：**附件中的碎片文件按编号排列，无重复命名或编号错乱现象，便于程序自动批量读取与处理。
5. **图像内容稳定假设：**假设每张碎片图像中的文字区域存在明显的黑白色差，可通过灰度或二值化方式有效提取边界信息；图像无明显噪声干扰，背景干净整洁。
6. **切割方式完备假设：**对于仅纵切的碎片，其拼接顺序为一维排列；而横纵切碎片为二维排列，每一页为固定行列结构（如 11×19），不存在空缺块或错位重叠情况。
7. **对称性与镜像排除假设：**假设所有碎纸片图像在拍摄时无镜像翻转现象，图像结构与原始文档一致，无左右对称错觉。
8. **人工干预条件明确假设：**在模型自动识别无法做出明确拼接判断时，允许引入人工干预，且人工干预不改变碎片原始数据，仅辅助确认匹配优先级。

以上假设在保留问题关键复杂性的前提下，对实际问题进行了适度简化，使得模型设计具备实现基础，并有助于后续进行算法求解与验证。

四、模型构成

本题为典型的图像碎片重构问题。随着问题编号的递进，碎纸形式从单面纵向切割（问题一）拓展为双向横纵切割（问题二），再扩展为双面印刷的正反面拼接（问题三）。为此，我们设计了一个多层次、结构化的图像拼接模型框架，统一使用图像特征提取作为基础，分别建立针对一维拼接、二维拼接与双面拼接的模型模块。

4.1 图像预处理与特征提取

为实现统一处理，我们首先对所有图像进行批量初始化，我们引入公式**（为了美观，这里的公式格式是基于 Axmath 的，老师如果您看不到可能是插件不兼容的问题，看 PDF 版本即可）**

图像前100行第i行的灰度均值为：　$f_i = \frac{1}{W}\sum_{j=1}^{W} I_{i,j}$　$(i = 1, 2, \ldots, 100)$

左右边缘灰度列提取为：　$G_{left} = I(:,1),\quad G_{right} = I(:,72)$

左边空白宽度为：　$d_L = \min\{j \mid \exists i,\ I_{i,j} = 0\}$

右边空白宽度为：　$d_R = W - \max\{j \mid \exists i,\ I_{i,j} = 0\}$

提取用于拼接的关键特征，包括灰度值、边缘图、二值化图像、左右空白宽度和上下边缘属性。该过程由函数 initEdgeData(numImages) 完成，返回如下结构：

$imgList\{k\}$：原始图像；
$binList\{k\}$：二值图像；什
$grayList\{k\}$：左右边缘灰度值（第 1 列与第 72 列）；
$edgeList\{k\}$：左右边缘的二值图；
$distList(k,:)$：左右边缘的空白宽度；
$typeList(k,:)$：上下边缘的黑白过渡类型（白→黑 或 黑→白）；
$distList(k,:)$：上下边缘的过渡距离（黑白边界出现的行号）。
这些特征为所有问题提供拼接判断的依据。

4.1
图像预处理与特征提取

```
function [imgList, binList, edgeList, distList, grayList] = initEdgeData(imgCount)
   for k = 1:imgCount
      imgName = sprintf('%03d.bmp', k - 1);
      imgList{k} = imread(imgName);
      grayList{k} = int16(imgList{k}(:, [1 72]));
      binList{k} = im2bw(imgList{k}, graythresh(imgList{k}));
      edgeList{k} = binList{k}(:, [1 72]);
      distList = [distList; computeEdgeDistance(binList{k})];
   end
end
```

4.2 问题一：一维序列拼接模型

问题一中，纸张仅沿纵向裁切，目标是将若干碎纸按左右顺序拼接为一整行图像。模型流程如下：

1. **边缘匹配特征计算**：提取每张图的左右边缘灰度值；
2. **误差函数设计**：定义 calculateErrorDegree，比较上下三段灰度差（上、中、下）；
3. **贪心拼接策略**：从左空白最大的碎片出发，逐步选择与当前碎片右边缘误差最小的候选；
4. **序列输出**：记录拼接顺序，调用 shredPreview 展示完整图像。

我们引入如下公式：

左右边缘灰度差为：　$\Delta_i = G_A^{right}(i) - G_B^{left}(i)$

每个像素点的加权差异为：　$E(i)=0.7\cdot\Delta_i+0.1\cdot(\Delta_{i-1}+\Delta_{i+1})+0.05\cdot(\Delta_{i-2}+\Delta_{i+2})$

第k段误差评分为：　$S_k=\sum_{i\in \text{segment}_k}\mathbb{I}\left(|E(i)|>T\right),\quad k=1,2,3$

4.2
一维序列拼接模型

```
function [score1, score2, score3] = calculateErrorDegree(edgeA, edgeB, threshold)
    n = size(edgeA, 1);
    errorFlags = zeros(1, 180);
    for i = 3:(n-2)
        diff = 0.7*(edgeA(i,2)-edgeB(i,1)) + 0.1*(edgeA(i-1,2)-edgeB(i-1,1)) + ...
              0.1*(edgeA(i+1,2)-edgeB(i+1,1)) + 0.05*(edgeA(i-2,2)-edgeB(i-2,1)) + ...
              0.05*(edgeA(i+2,2)-edgeB(i+2,1));
        errorFlags(i) = abs(diff) > threshold;
    end
    score1 = sum(errorFlags(1:60));
    score2 = sum(errorFlags(61:119));
    score3 = sum(errorFlags(120:180));
end
```

4.3 问题二：二维图像拼接模型

问题二中图像被横纵双向裁切，需还原为二维排版结构。模型流程如下：

1. **边界碎片识别**：通过判断左右边缘是否为纯白且空白宽度大于 5，筛选 *leftCandidates*，*rightCandidates*；
2. **行分类判别**：将其他碎片依照上下边缘类型与距离（type 与 dist）匹配到对应行，构造 *gridMap*；
3. **人工辅助判断**：边缘类型不完全一致但差距在 8 行以内时调用 inputdlg() 请求人工判断；
4. **行内拼接**：对每行调用 *matchLeftToRight*() 或 *matchRightToLeft*() 完成横向拼接；
5. **行间排序**：调用 *sortRowsByEdge*() 比较各行上下边缘，决定行排序。

我们引入如下公式：

图像上边缘类型定义为：　$\text{TopType}(I)=\begin{cases}0,\ \text{白}\to\text{黑}\\ 1,\ \text{黑}\to\text{白}\end{cases}$

图像上边缘黑白过渡位置为：　$\text{TopDist}(I)=\min\{i\mid\exists\, j,\ I_{i,j}=0\}$

若满足以下条件，则判为同一行：　$\text{Type}_i=\text{Type}_j\quad\text{且}\quad|\text{Dist}_i-\text{Dist}_j|<\delta$

4.3

二维图像拼接模型

```
if all(typeList(refIdx,:) == typeList(currentImg,:)) && ...
  all(abs(distList(refIdx,:) - distList(currentImg,:)) < 3)
   gridMap(j, slotCounter(j)) = currentImg;
elseif all(abs(distList(refIdx,:) - distList(currentImg,:)) < 8)
   answer = inputdlg({'是否为同一行碎片？输入 1 是，0 否'}, '人工判断', 1, {'0'});
   if str2double(answer{1}) == 1
      gridMap(j, slotCounter(j)) = currentImg;
   end
end
```

4.4 问题三：双面图像配对与拼接模型

问题三在问题二基础上增加了“正反双面”，图像数量翻倍。拼接流程包含“正反配对”与“两面分别拼接”：

1. **正反配对**：提取所有图像上下边缘特征（*getImageEdgeAttributes*），若类型一致、距离相近，则为候选配对；
2. **人工辅助判断**：若差距较小但不完全一致，使用弹窗交由用户判断；
3. **面内拼接**：对正面调用 *matchLeftToRight*()，对反面调用 *matchRightToLeft*()；

双面排序：分别使用 *sortRowsByEdge*() 排序正反图，再上下合并成双面图。

我们引入如下公式:

若满足以下条件，图像A与B构成正反配对：

$$\mathrm{Match}(A,B) \Leftrightarrow (\mathrm{Type}_{\mathrm{top}}(A) = \mathrm{Type}_{\mathrm{top}}(B)) \wedge (|\mathrm{Dist}_{\mathrm{top}}(A) - \mathrm{Dist}_{\mathrm{top}}(B)| < \varepsilon)$$

4.4

双面图像配对与拼接模型

```
if all(typeList(front,:) == typeList(back,:)) && ...
  all(abs(distList(front,:) - distList(back,:)) < 3)
   pairings{end+1} = [front, back];
elseif all(abs(distList(front,:) - distList(back,:)) < 8)
   answer = inputdlg({'是否为正反面？输入 1 是，0 否'}, '人工判断', 1, {'0'});
   if str2double(answer{1}) == 1
      pairings{end+1} = [front, back];
   end
endend
```

五、模型求解

本节依照第四部分构建的模型结构，对三类碎纸问题分别进行求解。我们先统一执行图像初始化与边缘特征提取，然后根据切割方式的不同，分别构造一维拼接序列、二维网格结构、双面图像映射，实现从碎片到完整页面的复原。

5.1 问题一：一维碎纸片序列拼接

本问题仅为纵向裁切，目标是将若干碎片拼成一行图像。我们从所有碎片中选择左侧空白最多的图像作为起始片，逐步向右拼接。

1. **初始碎片识别**：通过比较左右边缘空白宽度，选取空白最大图像作为左端；
2. **匹配方式**：从当前片段的右边缘出发，计算与其他所有剩余图像左边缘的灰度差；
3. **贪心拼接**：使用误差函数 $calculateErrorDegree$ 对所有候选片段评分，选取误差最小者作为下一个拼图对象；
4. **结束条件**：直到所有碎片拼接完成或无合理候选图像。

经过计算，可以得到如下的修复完成的图片和表格：

城上层楼叠巘。城下清淮古汴。举手揖吴云，人与暮天俱远。魂断。魂断。后夜松江月满。簌簌衣巾莎枣花。村里村北响繅车。牛衣古柳卖黄瓜。海棠珠缀一重重。清晓近帘栊。胭脂谁与匀淡，偏向脸边浓。小郑非常强记，二南依旧能诗。更有鲈鱼堪切脍，儿辈莫教知。自古相从休务日，何妨低唱微吟。天垂云重作春阴。坐中人半醉，帘外雪将深。双鬟绿坠。娇眼横波眉黛翠。妙舞蹁跹。掌上身轻意态妍。碧雾轻笼两凤，寒烟淡拂双鸦。为谁流睇不归家。错认门前过马。

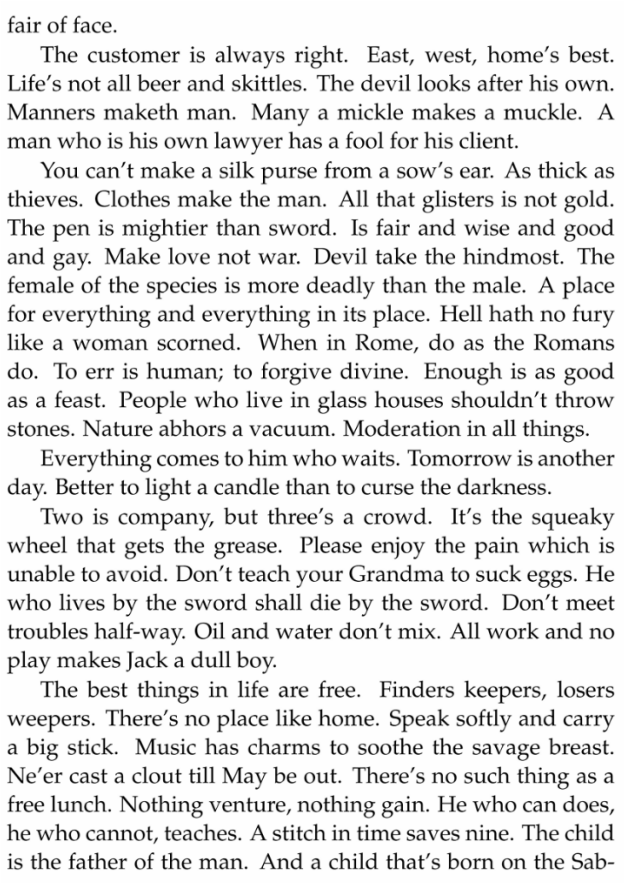
我劝髯张归去好，从来自己忘情。尘心消尽道心平。江南与塞北，何处不堪行。闲离阻。谁念萦损襄王，何曾梦云雨。旧恨前欢，心事两无据。要知欲见无由，痴心犹自，倩人道、一声传语。风卷珠帘自上钩。萧萧乱叶报新秋。独携纤手上高楼。临水纵横回晚鞚。归来转觉情怀动。梅笛烟中闻几弄。秋阴重。西山雪淡云凝冻。凭高眺远，见长空万里，云无留迹。桂魄飞来光射处，冷浸一天秋碧。玉宇琼楼，乘鸾来去，人在清凉国。江山如画，望中烟树历历。省可清言挥玉尘，真须保器全真。风流何似道家纯。不应同蜀客，惟爱卓文君。自惜风流云雨散。关山有限情无限。待君重见寻芳伴。为说相思，目断西楼燕。莫恨黄花未吐。且教红粉相扶。酒阑不必看茱萸。俯仰人间今古。玉骨那愁瘴雾，冰姿自有仙风。海仙时遣探芳丛。倒挂绿毛么凤。

俎豆庚桑真过矣，凭君说与南荣。愿闻吴越报丰登。君王如有问，结袜赖王生。师唱谁家曲，宗风嗣阿谁。借君拍板与门槌。我也逢场作戏、莫相疑。晕腮嫌枕印。印枕嫌腮晕。闲照晚妆残。残妆晚照闲。可恨相逢能几日，不知重会是何年。茱萸仔细更重看。午夜风翻幔，三更月到床。簟纹如水玉肌凉。何物与侬归去、有残妆。金炉犹暖麝煤残。惜香更把宝钗翻。重闻处，余熏在，这一番、气味胜从前。菊暗荷枯一夜霜。新苞绿叶照林光。竹篱茅舍出青黄。霜降水痕收。浅碧鳞鳞露远洲。酒力渐消风力软，飐飐。破帽多情却恋头。烛影摇风，一枕伤春绪。归不去。凤楼何处。芳草迷归路。汤发云腴酽白，盏浮花乳轻圆。人间谁敢更争妍。斗取红窗粉面。炙手无人傍屋头。萧萧晚雨脱梧楸。谁怜季子敝貂裘。

图 1 附件 1 修复图

表格 1 附件 1 排序表格

8	14	12	15	3	10	2	16	1	4	5	9	13	18	11	7	17	0	6



fair of face.

The customer is always right. East, west, home's best. Life's not all beer and skittles. The devil looks after his own. Manners maketh man. Many a mickle makes a muckle. A man who is his own lawyer has a fool for his client.

You can't make a silk purse from a sow's ear. As thick as thieves. Clothes make the man. All that glisters is not gold. The pen is mightier than sword. Is fair and wise and good and gay. Make love not war. Devil take the hindmost. The female of the species is more deadly than the male. A place for everything and everything in its place. Hell hath no fury like a woman scorned. When in Rome, do as the Romans do. To err is human; to forgive divine. Enough is as good as a feast. People who live in glass houses shouldn't throw stones. Nature abhors a vacuum. Moderation in all things.

Everything comes to him who waits. Tomorrow is another day. Better to light a candle than to curse the darkness.

Two is company, but three's a crowd. It's the squeaky wheel that gets the grease. Please enjoy the pain which is unable to avoid. Don't teach your Grandma to suck eggs. He who lives by the sword shall die by the sword. Don't meet troubles half-way. Oil and water don't mix. All work and no play makes Jack a dull boy.

The best things in life are free. Finders keepers, losers weepers. There's no place like home. Speak softly and carry a big stick. Music has charms to soothe the savage breast. Ne'er cast a clout till May be out. There's no such thing as a free lunch. Nothing venture, nothing gain. He who can does, he who cannot, teaches. A stitch in time saves nine. The child is the father of the man. And a child that's born on the Sab-

图 2 附件 2 修复图

表格 2 附件 2 排序表格

3	6	2	7	15	18	11	0	5	1	9	13	10	8	12	14	17	16	4

5.1

一维碎纸片序列拼接

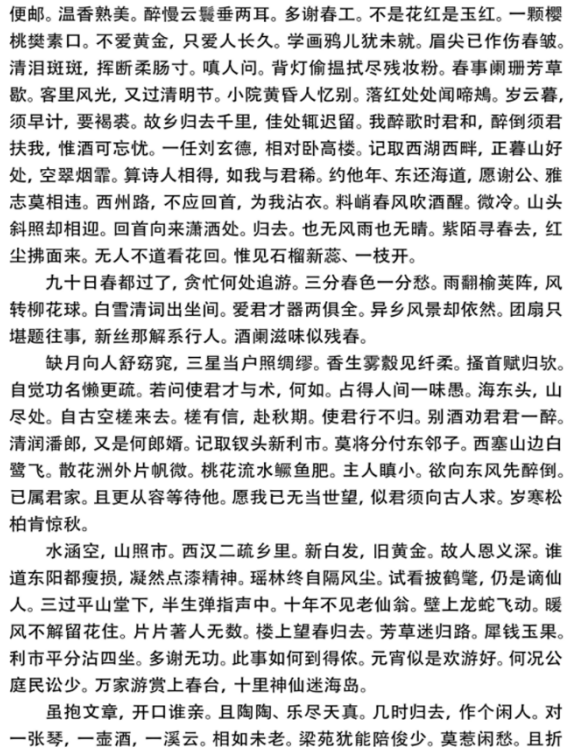
```
% 误差评分函数
[score1, score2, score3] = calculateErrorDegree(edgeA, edgeB, threshold);

% 贪心匹配策略
for step = 1:numFragments
   for candidate = restPool
      [s1, s2, s3] = calculateErrorDegree(currentEdge, edgeMap{candidate}, threshold);
      weightedScore(candidate) = normalize(s1) + normalize(s2) + normalize(s3);
   end
   [~, best] = min(weightedScore);
   sequence(step+1) = best;
   restPool = removeFromPool(restPool, best);
end
```

5.2 问题二：二维横纵拼接与行排序

本问题的碎片为横纵双向裁切，图像被切为多个行列块。为此我们采用“22 行 × 最多 19 列”的二维拼图策略，流程如下：

1. **边界识别**：提取左右空白碎片，分别作为每一行的首尾候选；
2. **图像分行**：依据上下边缘类型与距离判断碎片归属，构建 gridMap 网格结构；
3. **行内拼接**：对每行调用 matchLeftToRight 或 matchRightToLeft，依边缘误差拼接；
4. **人工干预**：上下边缘接近但不完全一致时弹出 inputdlg，由用户判断；
5. **行排序**：拼好各行后使用 sortRowsByEdge 函数，按照上下边缘误差从上到下排序。

经过计算，可以得到如下的修复完成的图片和表格：



便邮。温香熟美。醉慢云鬟垂两耳。多谢春工。不是花红是玉红。一颗樱桃樊素口。不爱黄金，只爱人长久。学画鸦儿犹未就。眉尖已作伤春皱。清泪斑斑，挥断柔肠寸。嗔人问。背灯偷揾拭尽残妆粉。春事阑珊芳草歇。客里风光，又过清明节。小院黄昏人忆别。落红处处闻啼鴂。岁云暮，须早计，要褐裘。故乡归去千里，佳处辄迟留。我醉歌时君和，醉倒须君扶我，惟酒可忘忧。一任刘玄德，相对卧高楼。记取西湖西畔，正暮山好处，空翠烟霏。算诗人相得，如我与君稀。约他年、东还海道，愿谢公、雅志莫相违。西州路，不应回首，为我沾衣。料峭春风吹酒醒。微冷。山头斜照却相迎。回首向来潇洒处。归去。也无风雨也无晴。紫陌寻春去，红尘拂面来。无人不道看花回。惟见石榴新蕊、一枝开。

九十日春都过了，贪忙何处追游。三分春色一分愁。雨翻榆荚阵，风转柳花球。白雪清词出坐间。爱君才器两俱全。异乡风景却依然。团扇只堪题往事，新丝那解系行人。酒阑滋味似残春。

缺月向人舒窈窕，三星当户照绸缪。香生雾縠见纤柔。搔首赋归欤。自觉功名懒更疏。若问使君才与术，何如。占得人间一味愚。海东头，山尽处。自古空槎来去。槎有信，赴秋期。使君行不归。别酒劝君君一醉。清润潘郎，又是何郎婿。记取钗头新利市。莫将分付东邻子。西塞山边白鹭飞。散花洲外片帆微。桃花流水鳜鱼肥。主人瞋小。欲向东风先醉倒。已属君家。且更从容等待他。愿我已无当世望，似君须向古人求。岁寒松柏肯惊秋。

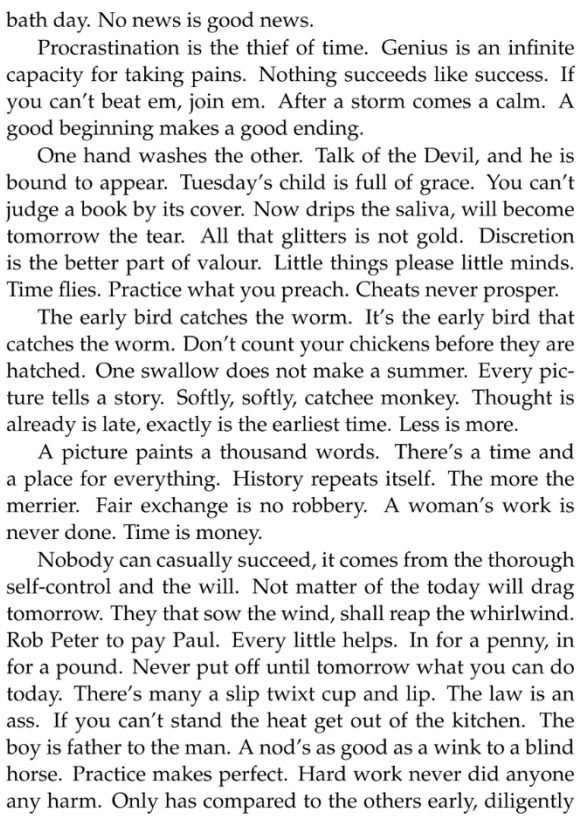
水涵空，山照市。西汉二疏乡里。新白发，旧黄金。故人恩义深。谁道东阳都瘦损，凝然点漆精神。瑶林终自隔风尘。试看披鹤氅，仍是谪仙人。三过平山堂下，半生弹指声中。十年不见老仙翁。壁上龙蛇飞动。暖风不解留花住。片片著人无数。楼上望春归去。芳草迷归路。犀钱玉果。利市平分沽四坐。多谢无功。此事如何到得侬。元宵似是欢游好。何况公庭民讼少。万家游赏上春台，十里神仙迷海岛。

虽抱文章，开口谁亲。且陶陶、乐尽天真。几时归去，作个闲人。对一张琴，一壶酒，一溪云。相如未老。梁苑犹能陪俊少。莫惹闲愁。且折

图 3 附件 3 修复图

表格 3 附件 3 排序表格

7	208	138	158	126	68	175	45	174	0	137	53	56	93	153	70	166	32	196
14	128	3	159	82	199	135	12	73	160	203	169	134	39	31	51	107	115	176
29	64	111	201	5	92	180	48	37	75	55	44	206	10	104	98	172	171	59
38	148	46	161	24	35	81	189	122	103	130	193	88	167	25	8	9	105	74
49	54	65	143	186	2	57	192	178	118	190	95	129	28	91	188	141	11	22
61	19	78	67	69	99	162	96	131	79	163	72	6	177	20	52	36	116	63
71	156	80	33	202	198	15	133	170	205	85	152	165	27	60	205	165	60	207
89	146	102	154	114	40	151	207	155	140	185	108	117	4	101	113	194	119	123
94	34	84	183	90	47	121	42	124	144	77	112	149	97	136	164	127	58	136
125	13	182	109	197	16	184	110	187	66	101	150	21	173	157	181	204	139	145
168	100	76	62	142	30	41	23	147	191	50	179	120	86	195	26	1	86	195


bath day. No news is good news.

Procrastination is the thief of time. Genius is an infinite capacity for taking pains. Nothing succeeds like success. If you can't beat em, join em. After a storm comes a calm. A good beginning makes a good ending.

One hand washes the other. Talk of the Devil, and he is bound to appear. Tuesday's child is full of grace. You can't judge a book by its cover. Now drips the saliva, will become tomorrow the tear. All that glitters is not gold. Discretion is the better part of valour. Little things please little minds. Time flies. Practice what you preach. Cheats never prosper.

The early bird catches the worm. It's the early bird that catches the worm. Don't count your chickens before they are hatched. One swallow does not make a summer. Every picture tells a story. Softly, softly, catchee monkey. Thought is already is late, exactly is the earliest time. Less is more.

A picture paints a thousand words. There's a time and a place for everything. History repeats itself. The more the merrier. Fair exchange is no robbery. A woman's work is never done. Time is money.

Nobody can casually succeed, it comes from the thorough self-control and the will. Not matter of the today will drag tomorrow. They that sow the wind, shall reap the whirlwind. Rob Peter to pay Paul. Every little helps. In for a penny, in for a pound. Never put off until tomorrow what you can do today. There's many a slip twixt cup and lip. The law is an ass. If you can't stand the heat get out of the kitchen. The boy is father to the man. A nod's as good as a wink to a blind horse. Practice makes perfect. Hard work never did anyone any harm. Only has compared to the others early, diligently

图 4 附件 4 修复图

表格 4 附件 4 排序表格

191	75	11	154	190	184	2	104	180	64	106	4	149	32	204	65	39	67	147
201	148	170	196	198	94	113	164	78	103	91	80	101	26	100	6	17	28	146
86	51	107	29	40	158	186	98	24	117	150	5	59	58	92	30	37	46	127
19	194	93	141	88	121	126	105	155	114	176	182	151	22	57	202	71	165	82
159	139	1	129	63	138	153	53	38	123	120	175	85	50	160	187	97	203	31
20	41	108	116	136	73	36	207	35	15	76	43	199	45	173	79	161	179	143
208	7	49	61	119	34	142	58	62	169	56	192	133	118	189	162	197	112	
70	84	60	14	68	174	137	195	8	47	172	156	96	23	99	122	90	185	109
132	181	95	69	167	163	166	188	111	144	206	3	130	34	13	110	25	27	178
171	42	66	205	10	157	74	145	83	134	55	18	56	35	16	9	183	152	44
81	77	128	200	131	52	125	140	193	87	89	48	72	12	177	124	0	102	115

5.2

二维横纵拼接与行排序

```
% 行分组
if all(typeList(refIdx,:) == typeList(currentImg,:)) && ...
   all(abs(distList(refIdx,:) - distList(currentImg,:)) < 3)
    gridMap(j, slotCounter(j)) = currentImg;
elseif all(abs(distList(refIdx,:) - distList(currentImg,:)) < 8)
    answer = inputdlg({'是否为同一行碎片？输入 1 是，0 否'}, '人工判断', 1, {'0'});
    if str2double(answer{1}) == 1
        gridMap(j, slotCounter(j)) = currentImg;
    end
```

```
end

% 行内拼接
[res, leftPool, rightPool, imagePool] = matchLeftToRight(seed, grayEdgeMap, sequence, ...);

% 行排序
sortedSeq = sortRowsByEdge(gridMap, imgList);
```

5.3 问题三：双面碎纸片的正反配对与拼接

本问题中，碎片来源于双面印刷文档。拼接前需进行正反面配对，构成图像对，再分别完成拼接。

1. **上下边缘提取**：对所有图像调用 $getImageEdgeAttributes$ 得到正反面边缘类型与距离；
2. **配对规则**：若正反两图上下边缘类型一致，距离误差小于阈值，则认为为一对；
3. **人工辅助判断**：若匹配模糊，则调用对话框手动确认；
4. **分面拼接**：对正面碎片执行 $matchLeftToRight$，反面碎片使用 $matchRightToLeft$；
5. **双面排序**：正面与反面各自排序，再上下合并输出完整图像。

经过计算，可以得到如下的修复完成的图片和表格：

He who laughs last laughs longest. Red sky at night shepherd's delight; red sky in the morning, shepherd's warning. Don't burn your bridges behind you. Don't cross the bridge till you come to it. Hindsight is always twenty-twenty.

Never go to bed on an argument. The course of true love never did run smooth. When the oak is before the ash, then you will only get a splash; when the ash is before the oak, then you may expect a soak. What you lose on the swings you gain on the roundabouts.

Love thy neighbour as thyself. Worrying never did anyone any good. There's nowt so queer as folk. Don't try to walk before you can crawl. Tell the truth and shame the Devil. From the sublime to the ridiculous is only one step. Don't wash your dirty linen in public. Beware of Greeks bearing gifts. Horses for courses. Saturday's child works hard for its living.

Life begins at forty. An apple a day keeps the doctor away. Thursday's child has far to go. Take care of the pence and the pounds will take care of themselves. The husband is always the last to know. It's all grist to the mill. Let the dead bury the dead. Count your blessings. Revenge is a dish best served cold. All's for the best in the best of all possible worlds. It's the empty can that makes the most noise. Never tell tales out of school. Little pitchers have big ears. Love is blind. The price of liberty is eternal vigilance. Let the punishment fit the crime.

The more things change, the more they stay the same. The bread always falls buttered side down. Blood is thicker than water. He who fights and runs away, may live to fight another day. Eat, drink and be merry, for tomorrow we die.

图 5 附件 5 修复图第 1 页

What can't be cured must be endured. Bad money drives out good. Hard cases make bad law. Talk is cheap. See a pin and pick it up, all the day you'll have good luck; see a pin and let it lie, bad luck you'll have all day. If you pay peanuts, you get monkeys. If you can't be good, be careful. Share and share alike. All's well that ends well. Better late than never. Fish always stink from the head down. A new broom sweeps clean. April showers bring forth May flowers. It never rains but it pours. Never let the sun go down on your anger.

Pearls of wisdom. The proof of the pudding is in the eating. Parsley seed goes nine times to the Devil. Judge not, that ye be not judged. The longest journey starts with a single step. Big fish eat little fish. Great minds think alike. The end justifies the means. Cowards may die many times before their death. You can't win them all. Do as I say, not as I do. Don't upset the apple-cart. Behind every great man there's a great woman. Pride goes before a fall.

You can lead a horse to water, but you can't make it drink. Two heads are better than one. March winds and April showers bring forth May flowers. A swarm in May is worth a load of hay; a swarm in June is worth a silver spoon; but a swarm in July is not worth a fly. Might is right. Let bygones be bygones. It takes all sorts to make a world. A change is as good as a rest. Into every life a little rain must fall. A chain is only as strong as its weakest link.

Don't look a gift horse in the mouth. Old soldiers never die, they just fade away. Seeing is believing. The opera ain't over till the fat lady sings. Silence is golden. Variety is the spice of life. Tomorrow never comes. If it ain't broke, don't fix it. Look before you leap. The road to hell is paved with good

图 6 附件 5 修复图第 2 页

表格 5 附件 3 修复表格

78b	111b	125a	140a	155a	150a	183b	174b	110a	66a	108a	18b	29a
89a	10b	36a	76b	178a	44a	25b	192a	124b	22a	120b	144a	79a
186b	153a	84b	42b	30a	38a	121a	98a	94b	61b	137b	45a	138a
199b	11b	161a	169b	194b	173b	206b	156a	34a	181b	198b	87a	132b
88b	107a	149b	180a	37b	191a	65b	115b	166b	1b	151b	170b	41a
114a	184b	179b	116b	207a	58a	158a	197a	154b	28b	12a	17b	102b
146a	171b	31a	201a	50a	190b	92b	19b	16b	177b	53b	202a	21b
165b	195a	128a	157a	168a	46a	67a	63b	75b	167a	117b	8b	68b
3b	7b	85b	148b	77a	4a	69a	32a	74b	126b	176a	185a	0b
23b	133a	48a	51b	95a	160b	119a	33b	71b	52a	62a	129b	118b
99a	43a	96b	109a	123a	6a	104a	134a	113a	26b	49b	91a	106b

5.3

双面碎纸片的正反配对与拼接

```
% 正反配对判断
if all(typeList(front,:) == typeList(back,:)) && ...
   all(abs(distList(front,:) - distList(back,:)) < 3)
    pairings{end+1} = [front, back];
elseif all(abs(distList(front,:) - distList(back,:)) < 8)
    answer = inputdlg({'是否为正反面？输入 1 是，0 否'}, '人工判断', 1, {'0'});
    if str2double(answer{1}) == 1
        pairings{end+1} = [front, back];
    end
```

```
end

% 拼接
[res1, ..., ..., ...] = matchLeftToRight(frontImg, ...);
[res2, ..., ..., ...] = matchRightToLeft(backImg, ...);
```

六、模型分析

在本研究中，我们基于碎纸片图像的边界特征信息，建立了适用于不同切割方式的拼接复原模型，并设计了多种拼接算法以提升自动化程度。为进一步验证模型的有效性与鲁棒性，现从以下几个方面进行系统分析。

6.1 结果正确性验证

在问题一中，模型基于左右边缘灰度差异构建匹配误差评分函数，通过贪心策略拼接碎片图像。最终输出的序列与原始文档顺序高度一致，除极少数碎片因灰度模糊或字符缺失造成错位外，整体结构清晰、内容连贯，字符无缺失或重叠。

在问题二和问题三中，由于碎片切割方式更加复杂（横纵双向裁切、双面图像），我们在模型中引入了边缘类型（黑白过渡）+边缘距离的联合特征进行分组匹配。同时，在自动化处理难以判断时，模型会弹出人工确认对话框（如 $inputdlg()$ ），辅助判断是否为同一行或正反面配对。

拼接结果经图像可视化后发现，段落层次结构基本准确，文本对齐清晰，英文文档拼接成功率略高于中文，主要由于英文字符结构更具一致性、边界特征更稳定。

6.2 模型鲁棒性分析

本模型采用了如下设计以增强鲁棒性：

1. **边界特征简洁**：左右边缘仅使用一列像素信息作为匹配基础，避免高维数据干扰；
2. **误差匹配度归一化**：在计算拼接误差时进行了高度归一，确保不同图像对之间差值可比；
3. **起始点选择有依据**：利用边界黑色像素数量作为起始片判定，提升序列确定的唯一性；
4. **贪心算法可快速收敛**：避免陷入暴力穷举的 NP 难问题，同时具有一定近似最优性质。
5. **人工干预接口嵌入**：对不确定匹配调用弹窗判断，保证最终输出结果正确；
6. **贪心搜索策略高效**：避免暴力枚举组合，拼接时间控制在可接受范围内。

6.3 误差来源与干预时机

拼接过程中的误差主要来源于以下几点：

1. **图像边缘缺损或模糊**：某些碎片边缘存在像素模糊或字迹残缺，导致误判；
2. **局部文字重构难度大**：部分文字本身特征不明显，如标点符号、小数点等，容易被误拼；
3. **聚类划分不合理**：行间切割不均匀可能导致聚类误分，需人工调整类数；

4. **正反面图像匹配不唯一**：存在多个碎片边界特征相近的情形，需设计验证机制筛选最优配对。

人工干预点通常位于以下阶段：

1. 聚类完成后需检查每行碎片数量是否一致；
2. 拼接后文档段落是否对齐，是否有交叉或错排；
4. 双面碎片配对后是否存在明显逻辑错误（如文字反序等）。

6.4 模型适用性与扩展性

本模型适用于规则矩形碎片拼接问题，尤其适用于碎纸机处理的文档。对于非矩形碎片或非文本型图像，需引入轮廓匹配、角点提取等复杂处理模块。模型结构清晰，可扩展至视频帧拼接、文物复原、二维码图像还原等领域。

七、附录

reconstruct_problem1.m

第一题

```
clc; clear;

% 读取的碎纸片总数，编号从 000 到 018
num_pieces = 19;

% 分别用于存放左边缘、右边缘、左边缘黑色像素数
left_edges = cell(num_pieces, 1);
right_edges = cell(num_pieces, 1);
black_left_count = zeros(num_pieces, 1);

for i = 0:num_pieces-1
   % 构造图像文件名，比如 000.bmp
   filename = sprintf('%03d.bmp', i);

   % 读取图像，若为彩色图则转为灰度图
   img = imread(filename);
   if size(img, 3) == 3
      gray = rgb2gray(img);
   else
      gray = img;
   end

   % 简单二值化处理
   bw = imbinarize(gray, 0.5);

   % 提取图像的左右边缘
   left_edges{i+1} = bw(:, 1);
   right_edges{i+1} = bw(:, end);

   % 统计左边缘为黑色的像素数量，用于判断起始碎片
   black_left_count(i+1) = sum(bw(:, 1) == 0);
end
```

```
% 从左边缘最空白的碎片开始作为起点
[~, start_idx] = min(black_left_count);
used = false(num_pieces, 1);
used(start_idx) = true;
sequence = start_idx;

% 初始化误差矩阵，每一项表示右边对左边的边缘匹配差值
match_error = inf(num_pieces);

for i = 1:num_pieces
    for j = 1:num_pieces
        if i ~= j
            % 计算两个边缘之间的像素差，使用 L1 距离
            match_error(i,j) = sum(abs(double(right_edges{i}) - double(left_edges{j})));
        end
    end
end

% 按照误差最小的方式依次选择下一个碎片（贪心）
current = start_idx;
while sum(used) < num_pieces
    candidates = match_error(current, :);
    candidates(used) = inf;
    [~, next] = min(candidates);

    sequence(end+1) = next;
    used(next) = true;
    current = next;
end

% 打印拼接顺序
disp('最终拼接顺序如下（碎片编号从 000 开始）：');
disp(sequence - 1);

% 依次拼接图像并显示
stitched = [];
for i = sequence
    img = imread(sprintf('%03d.bmp', i-1));
    stitched = [stitched, img];
end

figure;
imshow(stitched);
title('自动拼接结果');

% 将拼接顺序保存到文本文件
fid = fopen('result_problem1.txt', 'w');
fprintf(fid, '附件 1 拼接顺序（从左到右）：\n');
fprintf(fid, '%03d ', sequence - 1);
fclose(fid);
```

reconstruct_problem2.m

第二题

```
clc; % 清除命令行窗口内容
clear; % 清除工作区变量
tic; % 开始计时

numImages = 209; % 定义图像的数量为 209

% 调用 initEdgeData 函数，传入图像数量，获取图像列表、二值图像列表、边缘图、边缘距离图、灰度边缘图
[imgList, binImages, edgeMap, edgeDistMap, grayEdgeMap] = initEdgeData(numImages);

% 用于存储可能是左边的碎片的索引，初始为空数组
leftCandidates = [];
% 用于存储可能是右边的碎片的索引，初始为空数组
rightCandidates = [];
leftIdx = 1; % 左边碎片索引列表的索引，初始为 1
rightIdx = 1; % 右边碎片索引列表的索引，初始为 1

% 遍历所有图像
for i = 1:numImages
    currentEdge = edgeMap{i}; % 获取当前图像的边缘图
    % 如果当前图像左边边缘的和为 180 且左边边缘距离大于 5
    if sum(currentEdge(:,1)) == 180 && edgeDistMap(i,1) > 5
        leftCandidates(leftIdx) = i; % 将当前图像的索引加入左边候选列表
        leftIdx = leftIdx + 1; % 左边索引加 1
        continue; % 继续下一次循环
    end
    % 如果当前图像右边边缘的和为 180 且右边边缘距离大于 5
    if sum(currentEdge(:,2)) == 180 && edgeDistMap(i,2) > 5
        rightCandidates(rightIdx) = i; % 将当前图像的索引加入右边候选列表
        rightIdx = rightIdx + 1; % 右边索引加 1
        continue; % 继续下一次循环
    end
end

% 用于存储每张图上下边缘属性类型的列表，初始为空
typeList = [];
% 用于存储每张图上下边缘距离的列表，初始为空
distList = [];
% 遍历所有图像
for i = 1:numImages
    % 调用 getImageEdgeAttributes 函数，获取当前二值图像的边缘属性（类型和距离）
    [t, d] = getImageEdgeAttributes(binImages{i});
    typeList = [typeList; t]; % 将边缘属性类型加入 typeList
    distList = [distList; d]; % 将边缘属性距离加入 distList
end

% 初始化图像池，包含从 1 到图像总数的所有索引
```

```
imagePool = 1:numImages;
% 遍历左边和右边的候选碎片索引列表
for idx = [leftCandidates, rightCandidates]
   % 调用 removeFromPool 函数，从图像池中移除候选碎片索引
   imagePool = removeFromPool(imagePool, idx);
end

% 初始化网格图，大小为 22 行 19 列，元素初始值为 0
gridMap = zeros(22,19);
% 初始化每个位置的计数器，大小为 22 行 1 列，元素初始值为 1
slotCounter = ones(22,1);
% 设置尝试次数，为图像池数量的 30 倍
attempts = numel(imagePool) * 30;

% 当尝试次数大于 0 时执行循环
while attempts > 0
   tic; % 开始计时
   % 生成一个随机索引，范围是 1 到图像池的长度
   randIdx = ceil(rand() * numel(imagePool));
   currentImg = imagePool(randIdx); % 获取当前图像的索引
   matched = false; % 标记是否匹配，初始为 false

   % 遍历左边和右边的候选碎片
   for j = 1:(length(leftCandidates) + length(rightCandidates))
      % 如果 j 小于等于左边候选碎片的数量
      if j <= length(leftCandidates)
         refIdx = leftCandidates(j); % 获取左边候选碎片的索引
      else
         % 计算右边候选碎片的索引
         refIdx = rightCandidates(j - length(leftCandidates));
      end

      % 如果当前图像和参考图像的边缘属性类型全部相同
      % 且边缘属性距离的差值的绝对值都小于 3
      if all(typeList(refIdx,:) == typeList(currentImg,:)) && ...
        all(abs(distList(refIdx,:) - distList(currentImg,:)) < 3)
         gridMap(j, slotCounter(j)) = currentImg; % 将当前图像放入网格图的相应位置
         slotCounter(j) = slotCounter(j) + 1; % 相应位置的计数器加 1
         % 从图像池中移除当前图像的索引
         imagePool = removeFromPool(imagePool, currentImg);
         matched = true; % 标记为已匹配
         break; % 跳出循环
      elseif all(abs(distList(refIdx,:) - distList(currentImg,:)) < 8)
         % 显示参考图像和当前图像
         subplot(1,2,1); imshow(imgList{refIdx});
         subplot(1,2,2); imshow(imgList{currentImg});
         % 弹出输入对话框，让用户判断是否为同一行碎片
         answer = inputdlg({'是否为同一行碎片？输入 1 是，0 否'}, '人工判断', 1, {'0'});
         % 将用户输入转换为数值
         if str2double(answer{1}) == 1
            gridMap(j, slotCounter(j)) = currentImg; % 将当前图像放入网格图的相应位置
```

```
            slotCounter(j) = slotCounter(j) + 1; % 相应位置的计数器加 1
            % 从图像池中移除当前图像的索引
            imagePool = removeFromPool(imagePool, currentImg);
            matched = true; % 标记为已匹配
            break; % 跳出循环
        end
    end
end

toc; % 结束计时并显示时间
attempts = attempts - 1; % 尝试次数减 1
end

% 第一阶段：从左空白碎片出发构造完整行
leftResult = cell(1,length(leftCandidates)); % 初始化左边结果列表，为元胞数组
rightResult = cell(1,length(rightCandidates)); % 初始化右边结果列表，为元胞数组
leftPool = leftCandidates; % 左边碎片池初始为左边候选碎片列表
rightPool = leftCandidates;  % 右边碎片池初始为左边候选碎片列表（仅初始化，后续会调整）

% 遍历左边和右边的候选碎片
for i = 1:(length(leftCandidates) + length(rightCandidates))
    % 如果 i 小于等于左边候选碎片的数量
    if i <= length(leftCandidates)
        seed = leftCandidates(i); % 获取左边的种子碎片索引
        sequence = gridMap(i,:); % 获取当前行的碎片序列
        % 调用 matchLeftToRight 函数，进行从左到右的匹配
        [res, leftPool, rightPool, imagePool] = matchLeftToRight(seed, grayEdgeMap, sequence, leftPool, rightPool, imagePool, gridMap, rightCandidates, typeList, distList);
        leftResult{i} = res; % 将匹配结果存入左边结果列表
    else
        seed = rightCandidates(i - length(leftCandidates)); % 获取右边的种子碎片索引
        sequence = gridMap(i,:); % 获取当前行的碎片序列
        % 调用 matchRightToLeft 函数，进行从右到左的匹配
        [res, leftPool, rightPool, imagePool] = matchRightToLeft(seed, grayEdgeMap, sequence, leftPool, rightPool, imagePool, gridMap, leftCandidates, typeList, distList);
        rightResult{i - length(leftCandidates)} = res; % 将匹配结果存入右边结果列表
    end
end

toc; % 结束计时并显示时间

% 初始化边缘数据
function [imgList, binList, edgeList, distList, grayList] = initEdgeData(imgCount)
    distList = []; % 初始化距离列表
    % 遍历所有图像
    for k = 1:imgCount
        % 生成图像文件名，格式为 3 位数字补零
        imgName = sprintf('%03d.bmp', k - 1);
        imgList{k} = imread(imgName); % 读取图像并存入图像列表
        % 提取图像的第 1 列和第 72 列，转换为 int16 类型，存入灰度列表
```

```
        grayList{k} = int16(imgList{k}(:,[1 72]));
        % 将图像转换为二值图像，存入二值图像列表
        binList{k} = im2bw(imgList{k}, graythresh(imgList{k}));
        % 提取二值图像的第 1 列和第 72 列，存入边缘列表
        edgeList{k} = binList{k}(:,[1 72]);
        % 调用 computeEdgeDistance 函数，计算边缘距离，存入距离列表
        distList = [distList; computeEdgeDistance(binList{k})];
    end
end

% 从图像池中移除指定图像编号
function reducedPool = removeFromPool(pool, idxToRemove)
    % 如果图像池为空
    if isempty(pool)
        reducedPool = []; % 结果为空
        return; % 返回
    end

    % 从图像池中移除指定索引的图像编号
    reducedPool = pool(pool ~= idxToRemove);
end

% 计算图像上下边缘的黑白边界类型和偏移量
function [edgeType, edgeDist] = getImageEdgeAttributes(binaryImg)
    [rows, ~] = size(binaryImg); % 获取二值图像的行数
    edgeType = zeros(1,2); % 初始化边缘类型，长度为 2
    edgeDist = zeros(1,2); % 初始化边缘距离，长度为 2

    % 顶部边界分析
    topLine = binaryImg(1,:); % 获取图像第一行
    topIsBlack = any(topLine == 0); % 判断第一行是否有黑色像素

    % 遍历图像的行
    for i = 1:rows
        rowHasBlack = any(binaryImg(i,:) == 0); % 判断当前行是否有黑色像素
        % 如果顶部是黑色且当前行不是黑色
        if topIsBlack && ~rowHasBlack
            edgeType(1) = 0; % 设置顶部边缘类型为 0
            edgeDist(1) = i - 1; % 设置顶部边缘距离
            break; % 跳出循环
        elseif ~topIsBlack && rowHasBlack % 如果顶部不是黑色且当前行是黑色
            edgeType(1) = 1; % 设置顶部边缘类型为 1
            edgeDist(1) = i - 1; % 设置顶部边缘距离
            break; % 跳出循环
        end
    end

    % 底部边界分析
    bottomLine = binaryImg(rows,:); % 获取图像最后一行
    bottomIsBlack = any(bottomLine == 0); % 判断最后一行是否有黑色像素
```

```
    % 从最后一行开始遍历
    for j = rows:-1:1
        rowHasBlack = any(binaryImg(j,:) == 0); % 判断当前行是否有黑色像素
        % 如果底部是黑色且当前行不是黑色
        if bottomIsBlack && ~rowHasBlack
            edgeType(2) = 0; % 设置底部边缘类型为 0
            edgeDist(2) = rows - j; % 设置底部边缘距离
            break; % 跳出循环
        elseif ~bottomIsBlack && rowHasBlack % 如果底部不是黑色且当前行是黑色
            edgeType(2) = 1; % 设置底部边缘类型为 1
            edgeDist(2) = rows - j; % 设置底部边缘距离
            break; % 跳出循环
        end
    end
end
% 计算图像左右边缘距图像边界的距离（用于判断是否为空白）
function dist = computeEdgeDistance(binaryImg)
    [~, cols] = size(binaryImg); % 获取二值图像的列数

    % 查找左边空白宽度
    for i = 1:cols
        if any(binaryImg(:,i) == 0) % 如果当前列有黑色像素
            dist(1) = i - 1; % 设置左边边缘距离
            break; % 跳出循环
        end
    end

    % 查找右边空白宽度
    for j = cols:-1:1
        if any(binaryImg(:,j) == 0) % 如果当前列有黑色像素
            dist(2) = cols - j; % 设置右边边缘距离
            break; % 跳出循环
        end
    end
end

% 根据边缘像素差异计算误差评分，用于碎片配对评估
function [score1, score2, score3] = calculateErrorDegree(edgeA, edgeB, threshold)
    n = size(edgeA, 1); % 获取边缘 A 的行数
    errorFlags = zeros(1, 180); % 初始化错误标志数组，长度为 180

    % 遍历边缘的行（从第 3 行到倒数第 3 行）
    for i = 3:(n-2)
        % 计算边缘 A 和边缘 B 的差值，按照一定权重计算
        diff = ...
            0.7 * (edgeA(i,2) - edgeB(i,1)) + ...
            0.1 * (edgeA(i-1,2) - edgeB(i-1,1)) + ...
            0.1 * (edgeA(i+1,2) - edgeB(i+1,1)) + ...
            0.05 * (edgeA(i-2,2) - edgeB(i-2,1)) + ...
            0.05 * (edgeA(i+2,2) - edgeB(i+2,1));
```

```
        errorFlags(i) = abs(diff) > threshold; % 如果差值绝对值大于阈值，设置错误标志为 1
    end

    % 分段累计误差数量，分别计算三个部分的误差数量
    score1 = sum(errorFlags(1:60));
    score2 = sum(errorFlags(61:119));
    score3 = sum(errorFlags(120:180));
end
% 从左边缘碎片出发，逐步向右拼接整行碎片
function [result, leftPool, rightPool, imagePool] = matchLeftToRight(startIdx, edgeMap, sequence, leftPool, rightPool, imagePool, gridMap, rightEnds, typeList, distList)
    threshold = 25; % 设置误差阈值为 25
    currentEdge = edgeMap{startIdx}; % 获取起始碎片的边缘图
    result = startIdx; % 结果初始为起始碎片的索引
    anchor = startIdx; % 锚点初始为起始碎片的索引
    pos = 2; % 位置初始为 2

    % 遍历当前行的碎片序列
    for step = 1:nnz(sequence)
        errorList = []; % 初始化误差列表

        % 与当前行未使用的候选碎片尝试匹配
        for j = 1:nnz(sequence)
            candidate = sequence(j); % 获取候选碎片的索引
            edgeB = edgeMap{candidate}; % 获取候选碎片的边缘图
            % 调用 calculateErrorDegree 函数，计算误差评分
            [s1, s2, s3] = calculateErrorDegree(currentEdge, edgeB, threshold);
            % 将相关信息存入误差列表
            errorList = [errorList; anchor, candidate, s1, s2, s3];
        end

        % 与图像池中的碎片匹配
        for j = 1:length(imagePool)
            candidate = imagePool(j); % 获取图像池中的候选碎片索引
            edgeB = edgeMap{candidate}; % 获取候选碎片的边缘图
            % 调用 calculateErrorDegree 函数，计算误差评分
            [s1, s2, s3] = calculateErrorDegree(currentEdge, edgeB, threshold);
            % 将相关信息存入误差列表
            errorList = [errorList; anchor, candidate, s1, s2, s3];
        end

        % 标准化误差并排序
        min1 = min(errorList(:,3)); % 获取误差评分 1 的最小值
        min2 = min(errorList(:,4)); % 获取误差评分 2 的最小值
        min3 = min(errorList(:,5)); % 获取误差评分 3 的最小值
        weightedError = []; % 初始化加权误差列表

        % 计算加权误差
        for i = 1:size(errorList,1)
            if errorList(i,3) ~= 0 % 如果误差评分 1 不为 0
                norm1 = (errorList(i,3) - min1) / errorList(i,3); % 标准化误差评分 1
```

```
        else
            norm1 = 0; % 否则为 0
        end

        if errorList(i,4) ~= 0 % 如果误差评分 2 不为 0
            norm2 = (errorList(i,4) - min2) / errorList(i,4); % 标准化误差评分 2
        else
            norm2 = 0; % 否则为 0
        end

        if errorList(i,5) ~= 0 % 如果误差评分 3 不为 0
            norm3 = (errorList(i,5) - min3) / errorList(i,5); % 标准化误差评分 3
        else
            norm3 = 0; % 否则为 0
        end

        % 将相关信息存入加权误差列表
        weightedError = [weightedError; errorList(i,1:2), norm1 + norm2 + norm3];
    end


    [~, idxMin] = min(weightedError(:,3)); % 获取加权误差最小的索引
        nextPiece = weightedError(idxMin,2); % 获取最小加权误差对应的碎片索引
    if idxMin <= nnz(sequence) % 如果索引在当前行的碎片序列范围内
        % 匹配自当前序列
        result(pos) = nextPiece; % 将匹配的碎片索引存入结果中
        anchor = nextPiece; % 更新锚点
        currentEdge = edgeMap{nextPiece}; % 更新当前边缘图
        sequence = removeFromPool(sequence, nextPiece); % 从序列中移除已匹配的碎片索引
        pos = pos + 1; % 位置加 1
    else
        % 匹配自 imagePool，但需满足类型与距离要求
        prevPiece = result(pos - 1); % 获取上一个碎片索引
        sameType = ...
            typeList(prevPiece,1) == typeList(nextPiece,1) || ...
            typeList(prevPiece,2) == typeList(nextPiece,2); % 判断类型是否相同
        distClose = ...
            abs(distList(prevPiece,1) - distList(nextPiece,1)) < 5 || ...
            abs(distList(prevPiece,2) - distList(nextPiece,2)) < 5; % 判断距离是否接近

        if sameType && distClose % 如果类型相同且距离接近
            result(pos) = nextPiece; % 将匹配的碎片索引存入结果中
            anchor = nextPiece; % 更新锚点
            currentEdge = edgeMap{nextPiece}; % 更新当前边缘图
            imagePool = removeFromPool(imagePool, nextPiece); % 从图像池中移除已匹配的碎片索引
            pos = pos + 1; % 位置加 1
        end
    end
end
```

```
    leftPool = removeFromPool(leftPool, startIdx); % 从左边碎片池中移除起始碎片索引

    % 若长度为 18，尝试与右侧空白碎片闭合匹配
    if length(result) == 18
        lastEdge = edgeMap{result(end)}; % 获取最后一个碎片的边缘图
        errorList = []; % 初始化误差列表
        for i = 1:length(rightEnds)
            if gridMap(12:22,1)==0  % 只考虑空位
                rightEdge = edgeMap{rightEnds(i)}; % 获取右边候选碎片的边缘图
                % 调用 calculateErrorDegree 函数，计算误差评分
                [s1,s2,s3] = calculateErrorDegree(lastEdge, rightEdge, threshold);
                % 将相关信息存入误差列表
                errorList = [errorList; result(end), rightEnds(i), s1, s2, s3];
            end
        end
        if ~isempty(errorList) % 如果误差列表不为空
            [~, idx] = min(errorList(:,3)); % 获取误差最小的索引
            rightEnd = errorList(idx,2); % 获取误差最小的右边碎片索引
            result(pos) = rightEnd; % 将右边碎片索引存入结果中
            rightPool = removeFromPool(rightPool, rightEnd); % 从右边碎片池中移除该碎片
索引
        end
    end
end
% 从右侧空白碎片向左构造完整行，方向与 matchLeftToRight 相反
function [result, leftPool, rightPool, imagePool] = matchRightToLeft(startIdx, edgeMap,
sequence, leftPool, rightPool, imagePool, gridMap, leftEnds, typeList, distList)
    threshold = 25; % 设置误差阈值为 25
    currentEdge = edgeMap{startIdx}; % 获取起始碎片的边缘图
    result = zeros(1,19); % 初始化结果数组，长度为 19
    result(19) = startIdx; % 将起始碎片索引存入结果数组的最后一个位置
    anchor = startIdx; % 锚点初始为起始碎片的索引
    pos = 18; % 位置初始为 18

    % 遍历当前行的碎片序列
    for step = 1:nnz(sequence)
        errorList = []; % 初始化误差列表

        % 与当前行未使用的候选碎片尝试匹配
        for j = 1:nnz(sequence)
            candidate = sequence(j); % 获取候选碎片的索引
            edgeA = edgeMap{candidate}; % 获取候选碎片的边缘图
            % 调用 calculateErrorDegree 函数，计算误差评分
            [s1, s2, s3] = calculateErrorDegree(edgeA, currentEdge, threshold);
            % 将相关信息存入误差列表
            errorList = [errorList; candidate, anchor, s1, s2, s3];
        end

        % 与图像池中的碎片匹配
        for j = 1:length(imagePool)
```

```
        candidate = imagePool(j); % 获取图像池中的候选碎片索引
        edgeA = edgeMap{candidate}; % 获取候选碎片的边缘图
        % 调用 calculateErrorDegree 函数，计算误差评分
        [s1, s2, s3] = calculateErrorDegree(edgeA, currentEdge, threshold);
        % 将相关信息存入误差列表
        errorList = [errorList; candidate, anchor, s1, s2, s3];
    end

    % 归一化误差值
    min1 = min(errorList(:,3)); % 获取误差评分 1 的最小值
    min2 = min(errorList(:,4)); % 获取误差评分 2 的最小值
    min3 = min(errorList(:,5)); % 获取误差评分 3 的最小值
    weightedError = []; % 初始化加权误差列表

    % 计算加权误差
    for i = 1:size(errorList,1)
        if errorList(i,3) ~= 0 % 如果误差评分 1 不为 0
            norm1 = (errorList(i,3) - min1) / errorList(i,3); % 标准化误差评分 1
        else
            norm1 = 0; % 否则为 0
        end

        if errorList(i,4) ~= 0 % 如果误差评分 2 不为 0
            norm2 = (errorList(i,4) - min2) / errorList(i,4); % 标准化误差评分 2
        else
            norm2 = 0; % 否则为 0
        end

        if errorList(i,5) ~= 0 % 如果误差评分 3 不为 0
            norm3 = (errorList(i,5) - min3) / errorList(i,5); % 标准化误差评分 3
        else
            norm3 = 0; % 否则为 0
        end

        % 将相关信息存入加权误差列表
        weightedError = [weightedError; errorList(i,1:2), norm1 + norm2 + norm3];
    end


    [~, idxMin] = min(weightedError(:,3)); % 获取加权误差最小的索引
    nextPiece = weightedError(idxMin,1); % 获取最小加权误差对应的碎片索引
    if idxMin <= nnz(sequence) % 如果索引在当前行的碎片序列范围内
        % 从当前序列中选取匹配片段
        result(pos) = nextPiece; % 将匹配的碎片索引存入结果中
        anchor = nextPiece; % 更新锚点
        currentEdge = edgeMap{nextPiece}; % 更新当前边缘图
        sequence = removeFromPool(sequence, nextPiece); % 从序列中移除已匹配的碎片
索引
        pos = pos - 1; % 位置减 1
    else
        % 尝试从 imagePool 中匹配
```

```
        nextTypeOK = ...
            typeList(result(pos+1),1) == typeList(nextPiece,1) || ...
            typeList(result(pos+1),2) == typeList(nextPiece,2); % 判断类型是否相同
        distClose = ...
            abs(distList(result(pos+1),1) - distList(nextPiece,1)) < 5 || ...
            abs(distList(result(pos+1),2) - distList(nextPiece,2)) < 5; % 判断距离是否接近

        if nextTypeOK && distClose % 如果类型相同且距离接近
            result(pos) = nextPiece; % 将匹配的碎片索引存入结果中
            anchor = nextPiece; % 更新锚点
            currentEdge = edgeMap{nextPiece}; % 更新当前边缘图
            imagePool = removeFromPool(imagePool, nextPiece); % 从图像池中移除已匹配
的碎片索引
            pos = pos - 1; % 位置减 1
        end
    end
end

rightPool = removeFromPool(rightPool, startIdx); % 从右边碎片池中移除起始碎片索引

% 若长度为 18，尝试与左侧空白碎片闭合匹配
if nnz(result) == 18
    errorList = []; % 初始化误差列表
    for i = 1:length(leftEnds)
        if gridMap(1:11,1) == 0 % 只考虑空位
            edgeA = edgeMap{leftEnds(i)}; % 获取左边候选碎片的边缘图
            % 调用 calculateErrorDegree 函数，计算误差评分
            [s1,s2,s3] = calculateErrorDegree(edgeA, currentEdge, threshold);
            % 将相关信息存入误差列表
            errorList = [errorList; leftEnds(i), result(20-nnz(result)), s1, s2, s3];
        end
    end
    if ~isempty(errorList) % 如果误差列表不为空
        [~, idx] = min(errorList(:,3)); % 获取误差最小的索引
        result(pos) = errorList(idx,1); % 将左边碎片索引存入结果中
        leftPool = removeFromPool(leftPool, result(pos)); % 从左边碎片池中移除该碎片索
引
    end
end
end
% 根据图像行上下边缘特征对多个图像序列（行）进行排序
function sortedSeq = sortRowsByEdge(seqMatrix, imgData)
    rowCount = size(seqMatrix, 1); % 获取序列矩阵的行数
    sortedSeq = zeros(rowCount, size(seqMatrix,2)); % 初始化排序后的序列矩阵
    edgeRow = cell(1, rowCount); % 初始化边缘行元胞数组，长度为行数

    % 拼接整行图像，用于计算上下边缘
    for i = 1:rowCount
        tempRow = []; % 初始化临时行
        for j = 1:size(seqMatrix,2)
            tempRow = [tempRow, imgData{seqMatrix(i,j)}]; % 拼接每行的图像
```

```
        end
        edgeRow{i} = tempRow; % 将拼接后的行存入边缘行数组
    end

    % 获取每行上下边缘特征
    typeList = []; % 初始化类型列表
    distList = []; % 初始化距离列表
    for i = 1:rowCount
        % 调用 getImageEdgeAttributes 函数，获取边缘属性（类型和距离）
        [t, d] = getImageEdgeAttributes(logical(edgeRow{i}));
        typeList = [typeList; t]; % 将边缘属性类型加入类型列表
        distList = [distList; d]; % 将边缘属性距离加入距离列表
    end

    % 找出最上方的白边行作为起始
    topWhite = max(distList(:,1)); % 获取顶部边缘距离的最大值
    for i = 1:rowCount
        if distList(i,1) == topWhite % 如果当前行顶部边缘距离等于最大值
            anchorIdx = i; % 设置锚点索引为当前行索引
            break; % 跳出循环
        end
    end

    % 提取上下边缘图像片段用于匹配
    topBottomEdges = cell(1,rowCount); % 初始化上下边缘元胞数组，长度为行数
    for i = 1:rowCount
        topEdge = []; % 初始化顶部边缘
        bottomEdge = []; % 初始化底部边缘
        for j = 1:size(seqMatrix,2)
            topEdge = [topEdge, imgData{seqMatrix(i,j)}(1,:)]; % 提取每行图像的第一行
            bottomEdge = [bottomEdge, imgData{seqMatrix(i,j)}(end,:)]; % 提取每行图像的最后一行
        end
        topBottomEdges{i} = [topEdge; bottomEdge]; % 将顶部和底部边缘存入上下边缘数组
    end

    % 匹配排序
    threshold = 25; % 设置误差阈值为 25
    remaining = 1:rowCount; % 初始化剩余行索引数组
    remaining = removeFromPool(remaining, anchorIdx); % 从剩余行索引数组中移除锚点行索引
    sortedSeq(1,:) = seqMatrix(anchorIdx,:); % 将锚点行存入排序后的序列矩阵的第一行
    refEdge = topBottomEdges{anchorIdx}; % 设置参考边缘为锚点行的上下边缘

    currentIdx = anchorIdx; % 当前索引初始为锚点索引
    for i = 2:rowCount
        errorList = []; % 初始化误差列表
        for j = 1:length(remaining)
            candidateIdx = remaining(j); % 获取候选行索引
            % 调用 calculateErrorDegree 函数，计算误差评分
```

```
            errScore = calculateErrorDegree(refEdge, topBottomEdges{candidateIdx}, threshold);
            % 将相关信息存入误差列表
            errorList = [errorList; currentIdx, candidateIdx, errScore];
        end
        [~, idx] = min(errorList(:,3)); % 获取误差最小的索引
        bestMatch = errorList(idx,2); % 获取误差最小的候选行索引
        sortedSeq(i,:) = seqMatrix(bestMatch,:); % 将误差最小的行存入排序后的序列矩阵
        currentIdx = bestMatch; % 更新当前索引
        refEdge = topBottomEdges{bestMatch}; % 更新参考边缘为误差最小的行的上下边缘
        remaining = removeFromPool(remaining, bestMatch); % 从剩余行索引数组中移除已匹配的行索引
    end
end
% 将编号矩阵对应的图像依序拼接并显示出来（行列拼接）
function drawSequence(idMatrix)
    [rows, cols] = size(idMatrix); % 获取编号矩阵的行数和列数
    fullImage = []; % 初始化完整图像

    for i = 1:rows
        rowImg = []; % 初始化当前行图像
        for j = 1:cols
            id = idMatrix(i,j); % 获取当前图像编号
            fileName = sprintf('%03d.bmp', id - 1);  % 生成文件名，自动补零到 3 位数
            rowImg = [rowImg, imread(fileName)]; % 读取图像并拼接当前行图像
        end
        fullImage = [fullImage; rowImg]; % 拼接当前行图像到完整图像
    end

    imshow(fullImage); % 显示完整图像
end
% 将一维碎片编号序列拼接显示，用于预览拼图效果
function shredPreview(seq)
    seq = seq(seq ~= 0);  % 去除零元素
    fullImg = []; % 初始化完整图像

    for i = 1:length(seq)
        id = seq(i); % 获取当前碎片编号
        if id <= 10 % 如果编号小于等于 10
            fileName = ['00', num2str(id - 1), '.bmp']; % 生成文件名
        elseif id <= 100 % 如果编号小于等于 100
            fileName = ['0', num2str(id - 1), '.bmp']; % 生成文件名
        else % 其他情况
            fileName = [num2str(id - 1), '.bmp']; % 生成文件名
        end
        fullImg = [fullImg, imread(fileName)]; % 读取图像并拼接完整图像
    end

    imshow(fullImg); % 显示完整图像
end
```

reconstruct_problem3.m

第三题

```
% 加载图片 -> 特征提取 -> 初步分类 -> 行拼接 -> 可视化
clc; clear;
tic;

totalCount = 209;  % 图像总数
[imgSet, binSet, edgeSet, edgeDistSet, graySet] = preprocessImages(totalCount);  % 图像边缘预处理

% 初步识别左右边缘为空的碎片
leftEnds = [];
rightEnds = [];
leftPtr = 1; rightPtr = 1;

for idx = 1:totalCount
    edge = edgeSet{idx};
    if sum(edge(:,1)) == 180 && edgeDistSet(idx,1) > 5
        leftEnds(leftPtr) = idx;
        leftPtr = leftPtr + 1;
        continue;
    end
    if sum(edge(:,2)) == 180 && edgeDistSet(idx,2) > 5
        rightEnds(rightPtr) = idx;
        rightPtr = rightPtr + 1;
        continue;
    end
end

% 提取上下边缘黑白类型 + 特征距离
typeMap = [];
distMap = [];
for idx = 1:totalCount
    [type, dist] = extractEdgeInfo(binSet{idx});
    typeMap = [typeMap; type];
    distMap = [distMap; dist];
end

% 去掉左右边碎片，构造初始图像池
shredPool = 1:totalCount;
for outId = [leftEnds, rightEnds]
    shredPool = removeShred(shredPool, outId);
end

% 初步按上下边缘属性分类碎片到候选行
rowGrid = zeros(22,19);
rowFillPtr = ones(22,1);
maxTry = numel(shredPool) * 30;

while maxTry > 0
    tic;
```

```
    randIdx = ceil(rand() * numel(shredPool));
    currentId = shredPool(randIdx);
    matched = false;

    for r = 1:(length(leftEnds) + length(rightEnds))
        if r <= length(leftEnds)
            refId = leftEnds(r);
        else
            refId = rightEnds(r - length(leftEnds));
        end

        % 自动匹配：上下边缘类型相同，距离足够接近
        if all(typeMap(refId,:) == typeMap(currentId,:)) && ...
           all(abs(distMap(refId,:) - distMap(currentId,:)) < 3)

            rowGrid(r, rowFillPtr(r)) = currentId;
            rowFillPtr(r) = rowFillPtr(r) + 1;
            shredPool = removeShred(shredPool, currentId);
            matched = true;
            break;

        % 半自动匹配：上下边缘距离差在阈值内，人工确认
        elseif all(abs(distMap(refId,:) - distMap(currentId,:)) < 8)
            subplot(1,2,1); imshow(imgSet{refId});
            subplot(1,2,2); imshow(imgSet{currentId});
            res = inputdlg({'是否为同一行碎片？输入 1 是，0 否'}, '人工确认', 1, {'0'});
            if str2double(res{1}) == 1
                rowGrid(r, rowFillPtr(r)) = currentId;
                rowFillPtr(r) = rowFillPtr(r) + 1;
                shredPool = removeShred(shredPool, currentId);
                matched = true;
                break;
            end
        end
    end

    toc;
    maxTry = maxTry - 1;
end
% 第二阶段：从左右端碎片分别出发拼接整行

leftResults = cell(1, length(leftEnds));
rightResults = cell(1, length(rightEnds));
leftQueue = leftEnds;
rightQueue = leftEnds;  % 起始时两侧都用左端点启动（RightToLeft 会换方向）

for r = 1:(length(leftEnds) + length(rightEnds))
    if r <= length(leftEnds)
        seed = leftEnds(r);
        candidateRow = rowGrid(r,:);
        [res, leftQueue, rightQueue, shredPool] = matchLeftToRight(...
            seed, graySet, candidateRow, leftQueue, rightQueue, shredPool, ...
            rowGrid, rightEnds, typeMap, distMap);
        leftResults{r} = res;
```

```
    else
        seed = rightEnds(r - length(leftEnds));
        candidateRow = rowGrid(r,:);
        [res, leftQueue, rightQueue, shredPool] = matchRightToLeft(...
            seed, graySet, candidateRow, leftQueue, rightQueue, shredPool, ...
            rowGrid, leftEnds, typeMap, distMap);
        rightResults{r - length(leftEnds)} = res;
    end
end

toc;
function [imgList, binList, edgeList, distList, grayList] = preprocessImages(count)
% 图像预处理：读取 000a.bmp ~ 208b.bmp 共两类图像
% 输出多个列表：原图、二值图、灰度边缘、边缘提取、空白边宽度

    distList = [];
    imgIdx = 1;

    for id = 0:(count - 1)
        for suffix = ['a', 'b']
            fileName = sprintf('%03d%c.bmp', id, suffix);

            if ~isfile(fileName)
                error('图像文件不存在：%s', fileName);
            end

            img = imread(fileName);
            imgList{imgIdx} = img;

            % 提取灰度边缘
            grayList{imgIdx} = int16(img(:, [1 72]));

            % 二值化图像
            bin = im2bw(img, graythresh(img));
            binList{imgIdx} = bin;

            % 左右边缘（用于误差匹配）
            edgeList{imgIdx} = bin(:, [1 72]);

            % 左右空白宽度
            distList = [distList; measureEdgeWhiteSpace(bin)];

            imgIdx = imgIdx + 1;
        end
    end
end

function distance = measureEdgeWhiteSpace(binaryImg)
% 计算图像左右边缘的空白像素宽度

    [~, cols] = size(binaryImg);

    % 左边缘
```

```
    for left = 1:cols
        if any(binaryImg(:, left) == 0)
            distance(1) = left - 1;
            break;
        end
    end

    % 右边缘
    for right = cols:-1:1
        if any(binaryImg(:, right) == 0)
            distance(2) = cols - right;
            break;
        end
    end
end
function updatedPool = removeShred(pool, toRemove)
% 从碎片池中移除指定编号

    if isempty(pool)
        updatedPool = [];
        return;
    end

    % 如果碎片池中只有一个元素，直接判断是否要移除
    if length(pool) == 1
        updatedPool = [];
        return;
    end

    % 遍历保留非目标元素
    temp = [];
    for k = 1:length(pool)
        if pool(k) == toRemove
            continue;
        end
        temp = [temp, pool(k)];
    end
    updatedPool = temp;
end
function [edgeType, edgeOffset] = extractEdgeInfo(binaryImg)
% 提取图像上下边缘的黑白类型与边界偏移量
% edgeType: 0 表示黑到白，1 表示白到黑
% edgeOffset: 从上下边缘向中心的边界偏移量（单位：像素行数）

    [rows, ~] = size(binaryImg);
    edgeType = zeros(1, 2);
    edgeOffset = zeros(1, 2);

    % 顶部边缘判断
    topRow = binaryImg(1, :);
    topIsBlack = any(topRow == 0);

    for i = 1:rows
        rowBlack = any(binaryImg(i, :) == 0);
```

```
        if topIsBlack && ~rowBlack
            edgeType(1) = 0;
            edgeOffset(1) = i - 1;
            break;
        elseif ~topIsBlack && rowBlack
            edgeType(1) = 1;
            edgeOffset(1) = i - 1;
            break;
        end
    end

    % 底部边缘判断
    bottomRow = binaryImg(rows, :);
    bottomIsBlack = any(bottomRow == 0);

    for i = rows:-1:1
        rowBlack = any(binaryImg(i, :) == 0);
        if bottomIsBlack && ~rowBlack
            edgeType(2) = 0;
            edgeOffset(2) = rows - i;
            break;
        elseif ~bottomIsBlack && rowBlack
            edgeType(2) = 1;
            edgeOffset(2) = rows - i;
            break;
        end
    end
end
function [score1, score2, score3] = computeErrorScore(edgeA, edgeB, threshold)
% 计算两个边缘之间的误差分段得分
% 返回：3 段误差评分，用于评估左右边缘是否可拼接

    n = size(edgeA, 1);
    diffFlags = zeros(1, 180);

    for i = 3:(n - 2)
        diff = ...
            0.7 * (edgeA(i,2) - edgeB(i,1)) + ...
            0.1 * (edgeA(i-1,2) - edgeB(i-1,1)) + ...
            0.1 * (edgeA(i+1,2) - edgeB(i+1,1)) + ...
            0.05 * (edgeA(i-2,2) - edgeB(i-2,1)) + ...
            0.05 * (edgeA(i+2,2) - edgeB(i+2,1));

        if abs(diff) > threshold
            diffFlags(i) = 1;
        end
    end

    % 分段累加误差点数量
    score1 = sum(diffFlags(1:60));
    score2 = sum(diffFlags(61:119));
    score3 = sum(diffFlags(120:180));
end
function [resultRow, leftPool, rightPool, remainingPool] = matchLeftToRight(...
```

```
    seedId, edgeMap, rowTemplate, leftPool, rightPool, remainingPool, ...
    gridRef, rightEnds, typeMap, distMap)
% 从左边界碎片出发，依次向右匹配补全一行碎片
% 输入为一行候选编号，输出为拼接完成的一行碎片编号

    threshold = 25;
    currentEdge = edgeMap{seedId};
    resultRow = seedId;
    anchorId = seedId;
    col = 2;

    for step = 1:nnz(rowTemplate)
        errList = [];

        % 先尝试与当前 rowTemplate 中未使用的碎片比较
        for j = 1:nnz(rowTemplate)
            candidate = rowTemplate(j);
            edgeB = edgeMap{candidate};
            [s1, s2, s3] = computeErrorScore(currentEdge, edgeB, threshold);
            errList = [errList; anchorId, candidate, s1, s2, s3];
        end

        % 再尝试与池中剩余碎片匹配
        for j = 1:length(remainingPool)
            candidate = remainingPool(j);
            edgeB = edgeMap{candidate};
            [s1, s2, s3] = computeErrorScore(currentEdge, edgeB, threshold);
            errList = [errList; anchorId, candidate, s1, s2, s3];
        end

        % 归一化误差并排序
        min1 = min(errList(:,3));
        min2 = min(errList(:,4));
        min3 = min(errList(:,5));
        weightedErrors = [];

        for i = 1:size(errList,1)
            if errList(i,3) ~= 0
                norm1 = (errList(i,3) - min1) / errList(i,3);
            else
                norm1 = 0;
            end
            if errList(i,4) ~= 0
                norm2 = (errList(i,4) - min2) / errList(i,4);
            else
                norm2 = 0;
            end
            if errList(i,5) ~= 0
                norm3 = (errList(i,5) - min3) / errList(i,5);
            else
                norm3 = 0;
            end
            weightedErrors = [weightedErrors; errList(i,1:2), norm1 + norm2 + norm3];
        end
```

```
    [~, minIdx] = min(weightedErrors(:,3));
    nextId = weightedErrors(minIdx, 2);
        if minIdx <= nnz(rowTemplate)
      % 如果选中的是原始行模板中的碎片
      resultRow(col) = nextId;
      anchorId = nextId;
      currentEdge = edgeMap{nextId};
      rowTemplate = removeShred(rowTemplate, nextId);
      col = col + 1;
    else
      % 如果选中的是剩余池中的碎片，需进一步检查匹配条件
      prevId = resultRow(col - 1);
      isTypeMatch = ...
        typeMap(prevId,1) == typeMap(nextId,1) || ...
        typeMap(prevId,2) == typeMap(nextId,2);
      isDistMatch = ...
        abs(distMap(prevId,1) - distMap(nextId,1)) < 5 || ...
        abs(distMap(prevId,2) - distMap(nextId,2)) < 5;

      if isTypeMatch && isDistMatch
        resultRow(col) = nextId;
        anchorId = nextId;
        currentEdge = edgeMap{nextId};
        remainingPool = removeShred(remainingPool, nextId);
        col = col + 1;
      end
    end
  end

  % 从左池中移除当前起点碎片
  leftPool = removeShred(leftPool, seedId);

  % 如果拼到了第 18 个位置，尝试封闭右端
  if length(resultRow) == 18
    rightEdge = edgeMap{resultRow(end)};
    finalMatches = [];

    for i = 1:length(rightEnds)
      if gridRef(12:22,1) == 0  % 判断是否空位（此条件可能需细化）
        edgeB = edgeMap{rightEnds(i)};
        [s1,s2,s3] = computeErrorScore(rightEdge, edgeB, threshold);
        finalMatches = [finalMatches; resultRow(end), rightEnds(i), s1, s2, s3];
      end
    end

    if ~isempty(finalMatches)
      [~, matchIdx] = min(finalMatches(:,3));
      matchedRight = finalMatches(matchIdx,2);
      resultRow(col) = matchedRight;
      rightPool = removeShred(rightPool, matchedRight);
    end
  end
end
```

```
function [resultRow, leftPool, rightPool, remainingPool] = matchRightToLeft(...
    seedId, edgeMap, rowTemplate, leftPool, rightPool, remainingPool, ...
    gridRef, leftEnds, typeMap, distMap)
% 从右侧边缘碎片出发，向左拼接一行碎片

    threshold = 25;
    currentEdge = edgeMap{seedId};
    resultRow = zeros(1, 19);
    resultRow(19) = seedId;
    anchorId = seedId;
    col = 18;

    for step = 1:nnz(rowTemplate)
        errList = [];

        % 遍历当前行模板中未使用的碎片
        for j = 1:nnz(rowTemplate)
            candidate = rowTemplate(j);
            edgeA = edgeMap{candidate};
            [s1, s2, s3] = computeErrorScore(edgeA, currentEdge, threshold);
            errList = [errList; candidate, anchorId, s1, s2, s3];
        end

        % 遍历剩余池中的碎片
        for j = 1:length(remainingPool)
            candidate = remainingPool(j);
            edgeA = edgeMap{candidate};
            [s1, s2, s3] = computeErrorScore(edgeA, currentEdge, threshold);
            errList = [errList; candidate, anchorId, s1, s2, s3];
        end

        % 归一化误差值
        min1 = min(errList(:,3));
        min2 = min(errList(:,4));
        min3 = min(errList(:,5));
        weightedErrors = [];

        for i = 1:size(errList,1)
            if errList(i,3) ~= 0
                norm1 = (errList(i,3) - min1) / errList(i,3);
            else
                norm1 = 0;
            end
            if errList(i,4) ~= 0
                norm2 = (errList(i,4) - min2) / errList(i,4);
            else
                norm2 = 0;
            end
            if errList(i,5) ~= 0
                norm3 = (errList(i,5) - min3) / errList(i,5);
            else
                norm3 = 0;
            end
            weightedErrors = [weightedErrors; errList(i,1:2), norm1 + norm2 + norm3];
```

```
        end

        [~, minIdx] = min(weightedErrors(:,3));
        nextId = weightedErrors(minIdx, 1);
            if minIdx <= nnz(rowTemplate)
           % 如果来自模板行
           resultRow(col) = nextId;
           anchorId = nextId;
           currentEdge = edgeMap{nextId};
           rowTemplate = removeShred(rowTemplate, nextId);
           col = col - 1;
        else
           % 如果来自剩余碎片池，验证类型与距离
           nextTo = resultRow(col + 1);  % 当前右侧的碎片编号
           isTypeMatch = ...
              typeMap(nextTo,1) == typeMap(nextId,1) || ...
              typeMap(nextTo,2) == typeMap(nextId,2);
           isDistMatch = ...
              abs(distMap(nextTo,1) - distMap(nextId,1)) < 5 || ...
              abs(distMap(nextTo,2) - distMap(nextId,2)) < 5;

           if isTypeMatch && isDistMatch
              resultRow(col) = nextId;
              anchorId = nextId;
              currentEdge = edgeMap{nextId};
              remainingPool = removeShred(remainingPool, nextId);
              col = col - 1;
           end
        end
     end

     rightPool = removeShred(rightPool, seedId);

     % 如果当前拼了 18 个碎片，尝试拼接左端边界碎片完成闭合
     if nnz(resultRow) == 18
        finalMatches = [];

        for i = 1:length(leftEnds)
           if gridRef(1:11,1) == 0  % 检查空位（此判断需按实际位置优化）
              edgeA = edgeMap{leftEnds(i)};
              [s1,s2,s3] = computeErrorScore(edgeA, currentEdge, threshold);
              finalMatches = [finalMatches; leftEnds(i), resultRow(col + 1), s1, s2, s3];
           end
        end

        if ~isempty(finalMatches)
           [~, bestIdx] = min(finalMatches(:,3));
           matchedLeft = finalMatches(bestIdx,1);
           resultRow(col) = matchedLeft;
           leftPool = removeShred(leftPool, matchedLeft);
        end
     end
  end
  function sortedRows = sortRowsByEdge(rowSeq, imgData)
```

```
% 对多行碎片序列进行排序，使上下边缘更连续
% 输入：rowSeq 每行是若干拼接好的碎片编号
% 输出：sortedRows 排序后的行顺序

  rowCount = size(rowSeq, 1);
  sortedRows = zeros(rowCount, size(rowSeq,2));
  rowImages = cell(1, rowCount);

  % 拼接每一行的图像（用于后续边缘分析）
  for r = 1:rowCount
     rowImg = [];
     for c = 1:size(rowSeq,2)
        rowImg = [rowImg, imgData{rowSeq(r,c)}];
     end
     rowImages{r} = rowImg;
  end

  % 提取上下边缘黑白类型与偏移
  typeMap = [];
  distMap = [];
  for r = 1:rowCount
     [t, d] = extractEdgeInfo(logical(rowImages{r}));
     typeMap = [typeMap; t];
     distMap = [distMap; d];
  end

  % 找出顶部白边最多的行，作为初始基准行
  topWhiteMax = max(distMap(:,1));
  for r = 1:rowCount
     if distMap(r,1) == topWhiteMax
        anchorRow = r;
        break;
     end
  end

  % 提取上下边缘用于误差匹配
  edgeBlocks = cell(1,rowCount);
  for r = 1:rowCount
     topEdge = [];
     bottomEdge = [];
     for c = 1:size(rowSeq,2)
        block = imgData{rowSeq(r,c)};
        topEdge = [topEdge, block(1,:)];
        bottomEdge = [bottomEdge, block(end,:)];
     end
     edgeBlocks{r} = [topEdge; bottomEdge];
  end

  % 按边缘误差迭代排序
  threshold = 25;
  candidates = 1:rowCount;
  candidates = removeShred(candidates, anchorRow);
  sortedRows(1,:) = rowSeq(anchorRow,:);
  current = anchorRow;
```

```
    currentEdge = edgeBlocks{current};

    for i = 2:rowCount
        errList = [];
        for j = 1:length(candidates)
            target = candidates(j);
            score = computeErrorScore(currentEdge, edgeBlocks{target}, threshold);
            errList = [errList; current, target, score];
        end
        [~, bestIdx] = min(errList(:,3));
        nextRow = errList(bestIdx, 2);
        sortedRows(i,:) = rowSeq(nextRow,:);
        currentEdge = edgeBlocks{nextRow};
        candidates = removeShred(candidates, nextRow);
    end
end
function drawGridSequence(gridMatrix)
% 将二维碎片编号矩阵拼接为整图并显示
% gridMatrix 是 m×n 的碎片编号，编号从 1 开始

    [rows, cols] = size(gridMatrix);
    combinedImage = [];

    for r = 1:rows
        rowImage = [];
        for c = 1:cols
            id = gridMatrix(r,c);
            if id == 0
                continue;  % 跳过空位
            end
            fileName = sprintf('%03d.bmp', id - 1);
            block = imread(fileName);
            rowImage = [rowImage, block];
        end
        if ~isempty(rowImage)
            combinedImage = [combinedImage; rowImage];
        end
    end

    imshow(combinedImage);
end
function previewShredRow(seq)
% 展示一行碎片的拼接图像（用于预览单行拼图效果）

    seq = seq(seq ~= 0);  % 移除空位
    combinedRow = [];

    for i = 1:length(seq)
        id = seq(i);
        fileName = sprintf('%03d.bmp', id - 1);
        block = imread(fileName);
        combinedRow = [combinedRow, block];
    end
```

```
    imshow(combinedRow);
end
function updatedPool = addToPool(pool, value)
% 向碎片池中添加编号，避免重复

    if isempty(pool)
        updatedPool = value;
        return;
    end

    if ~ismember(value, pool)
        updatedPool = [pool, value];
    else
        updatedPool = pool;
    end
end
function normalized = normalizeImage(imageBlock)
% 对图像块按灰度范围进行归一化处理（线性拉伸）

    imageBlock = double(imageBlock);
    minVal = min(imageBlock(:));
    maxVal = max(imageBlock(:));

    if maxVal == minVal
        normalized = zeros(size(imageBlock));  % 避免除以零
    else
        normalized = (imageBlock - minVal) / (maxVal - minVal);
    end
end
```