




# DSAAB PROJECT

## 一. 项目的基本结构

项目总共分为六个类：

 Get_path.java	2025/5/10 21:53	Java 源文件
 ImageReader.java	2025/5/10 19:18	Java 源文件
 ImageUploader.java	2025/5/10 21:10	Java 源文件
 LiveWire.java	2025/5/10 4:13	Java 源文件
 Main.java	2025/5/10 21:11	Java 源文件
 Pixel.java	2025/5/9 20:39	Java 源文件

1. ImageReader.java 负责图像输入预处理的核心模块，使用 Java 标准库 ImageIO 从本地读取图像文件。将彩色图像解析为三维数组 `int[height][width][3]`，表示 RGB 通道。支持将图像部分区域（默认 10x10 像素）打印输出以辅助调试。提供将 RGB 像素数组保存到 .txt 文件的接口 `saveImageArrayToFile()`，用于中间结果验证与调试。
2. Get\_path.java 是整个项目的核心类
  - ①负责从 ImageReader.java 中读取图像转化成的三维数组，对该三维数组进行一系列处理得到路径成本数组（该数组的每个元素是走过此位置所需要的成本）。
  - ②还负责根据从 LiveWire.java 得到的数据算出终点到起始点的最短路径。
  - ③并且在 Get\_path.java 中还有 `path_cooling` 和 `cursor_snap` 两个附加功能的代码。
3. LiveWire.java 实现了迪杰斯特拉算法，根据从 Get\_path.java 中得到的路径成本的数组和输入的起始点坐标得到：从该起始点出发，图像的每个像素点到该起始点最小成本的路径经过哪些像素和该成本是多少。
4. Pixel.java 中包含了一种数据结构，该数据结构包含了 x,y 坐标和从起始点该像素的成本，被 Get\_path.java 和 LiveWire.java 多次使用。该类继承了 `comparable` 接口，用于比较不同路径到该像素点的成本。
5. ImageUploader.java 是 JFrame 的子类，继承 JFrame 方法。在代码中，主要运用了 swing 类的代码实现 GUI（图形用户界面）；使用 File 得到图片路径，赋给 Image 相关类实现用户输入图片；用 Linklist 类储存得到的路径值；用 BufferedImage, Path2D 裁剪出截图；用 JFileChooser 和 File 保存生成的截图
6. Main.java main 函数的所在类，项目运行的起点。

## 二. 运用的数据结构

1. ImageReader.java :

①int[][][]: 表示 RGB 彩色图像数据。

②FileWriter: 输出调试文件, 结合 for 嵌套迭代进行有序访问像素矩阵。

2. Get\_path.java:

①Pixel[]: 数组中的每个元素将包含一个 Pixel 对象, 该对象记录了从起点到该像素的路径信息。例如, Pixel[100][100] 包含一个 Pixel 对象包括上一个像素的位置和路径成本。从而让终点不断寻找上一个像素, 直到找到起点。

②double[]: 用于对图像数据的处理

③StringBuilder[]: 用于存储终点到起点的最小成本路径。

3. LiveWire.java

①boolean[]: 用于保存数组中的每个元素是否被处理过。

②PriorityQueue<Pixel>, 运用了优先队列, 每次取出的都是路径成本最小的。大大减小了复杂度。

4. Pixel.java: 这个类就是 Pixel 这个数据结构, 包含像素的 x,y 坐标和每个像素到起点的成本。并且 Pixel.java 实现了 comparable 的接口, 用于该像素由不同路径到起点所需成本的比较。

### 三 . 性能优化的方法:

1. ImageReader.java:

在 saveImageArrayToFile() 中限制输出区域为 10x10, 防止大图调试造成性能阻塞或控制台溢出。

文件写出操作中判断输出路径是否存在, 避免 I/O 异常。

所有数组访问都预先计算长宽值, 避免重复 .length 调用。

2. LiveWire.java :

PriorityQueue<Pixel>, 运用了优先队列来得到最小路径成本的点, 从而减小了复杂度, 优化了性能。

3. ImageUploader.java: 对于 Get\_path 类得到的路径, 是 StringBuilder 类, 坐标间以“, ”作为分割的 String 数据。对于长度未知, 需要频繁读取队首数据和结尾数据和遍历数据, 我选用链表 (Linklist) 来存储数据, 并且对于临时链表每次使用结束都会清除数据, 节约空间。

### 四 . 项目特色:

1. ImageReader.java

①使用相对路径 resources/ 和输出路径 output/, 方便项目跨平台移植和部署。

②添加了自动创建文件夹的能力, 确保输出文件不会因路径不存在而中断。

2. Get\_path.java: 由于在描边的时候无法了解长方形图片的边界是不是边缘 (因为边界只有一边, 无法算出梯度), 且不清楚用户具体要求。例如用户可能截一下这半张脸, 但这半张脸是紧贴这个长方形图片的左边界的。所以为了优化整体体验, 我们小组采用

了“宁可信其有，不可信其无”的策略，我们将长方形图像的边界成本降低（倾向于长方形边界就是边缘），路径更加倾向于沿边界走。

截图时



可以看到，左边界被当成最低成本路径

截图后



3. ImageUploader.java:

- ①可选择路径读取图片与可选择路径存储图片，更贴近实际应用。
- ②当实现路径冷却和光标偏移时，会在 Jlabel 上显示，同时 `println()` 便于测试。

## 五 统计数据：本项目核心算法的复杂度

```
public Pixel[][] computePaths(int startX, int startY) {
    PriorityQueue<Pixel> activeList = new PriorityQueue<>();
    p[startX][startY]=null;
    g[startX][startY] = 0;
    activeList.add(new Pixel(startX, startY, 0));

    while (!activeList.isEmpty()) {
        Pixel q = activeList.poll();
        int x = q.x, y = q.y;
        if (processed[x][y]) continue;
        processed[x][y] = true;

        // 遍历 8 邻域
        for (int dy = -1; dy <= 1; dy++) {
```

```

        for (int dx = -1; dx <= 1; dx++) {
            if (dx == 0 && dy == 0) continue;
            int nx = x + dx, ny = y + dy;
            if (ny < 0 || ny >= height || nx < 0 || nx >= width)
continue;

            double newCost = g[x][y] + localCost(dx, dy, ny, nx);
            if (newCost < g[nx][ny]) {
                g[nx][ny] = newCost;
                p[nx][ny] = new Pixel(x, y, newCost);
                activeList.add(new Pixel(nx, ny, newCost));
            }
        }
    }
}
return p;
}

```

这个函数方法实现了迪杰斯特拉算法，用于计算从该起始点出发，图像的每个像素点到该起始点最小成本的路径经过哪些像素和该成本是多少

当一个像素从优先队列中移除时，它通过计算对所有尚未扩展的邻居的累积成本来扩展。在最坏情况下，一个像素的累积成本由它的 8 个邻居计算得出，从而导致  $N$  个像素的  $8N$  个累积成本计算。显然，并不是每个点在所有邻居扩展后都可以扩展。除了种子点外，任何具有累积成本的点都必须拥有至少一个已经扩展的邻居。因此，对于那些邻居，不需要重新计算累积成本。简而言之，可以证明最多只执行  $4N$  个累积成本计算，从而导致  $O(N)$  的算法。

## 二，如何运行代码

### 1，运行 Main 函数

```

package dsaab_project;

import java.util.*;

import dsaab_project.*;

public class Main {

    public static void main(String[] args) {

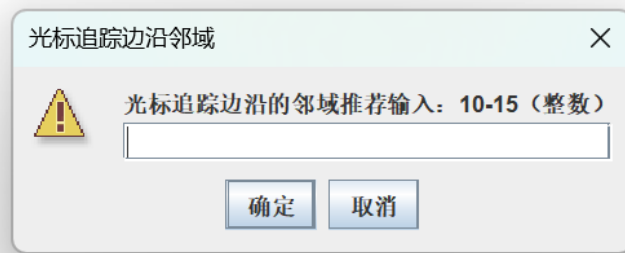
        ImageUploader a = new ImageUploader();

    }

}

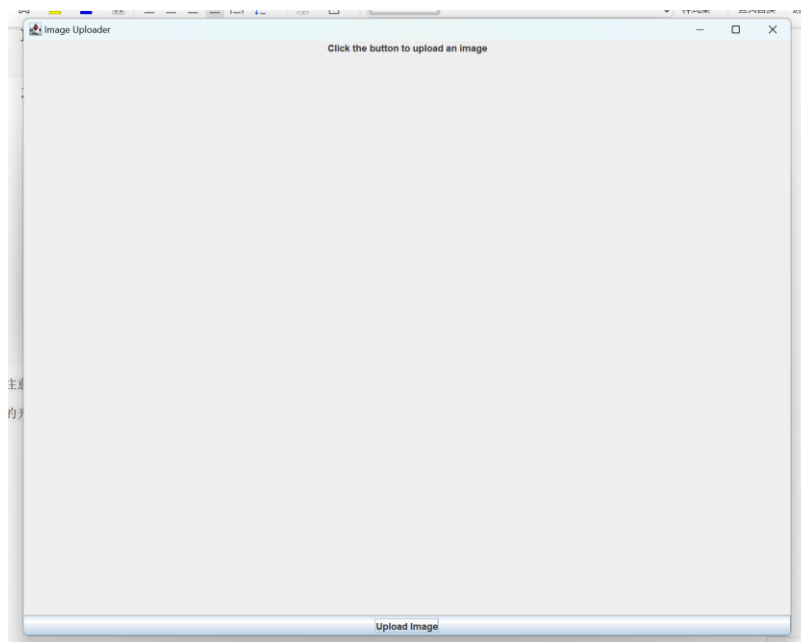
```

### 2，填入邻域 int 值



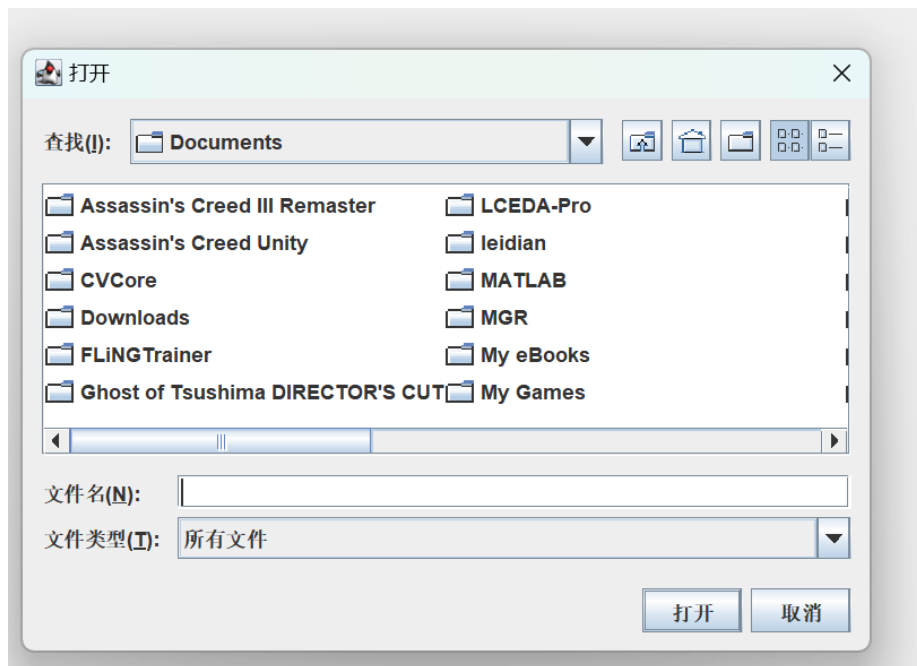
请注意，一定要填 int 值，因为像素的个数为整数。int 值的大小代表邻域的大小，代表边沿判断的范围。推荐输入 10-15 的某个值，经过我们的测试，在这个范围的光标追踪边沿方法效果最好。

3，点击按钮“upload image”



按钮在最下方，请点击。

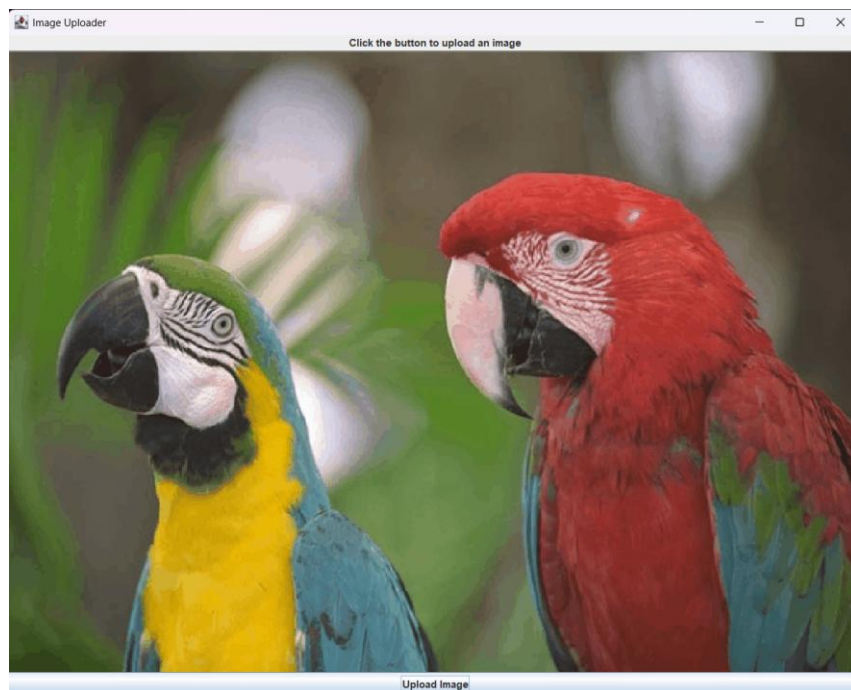
#### 4, 选择图片



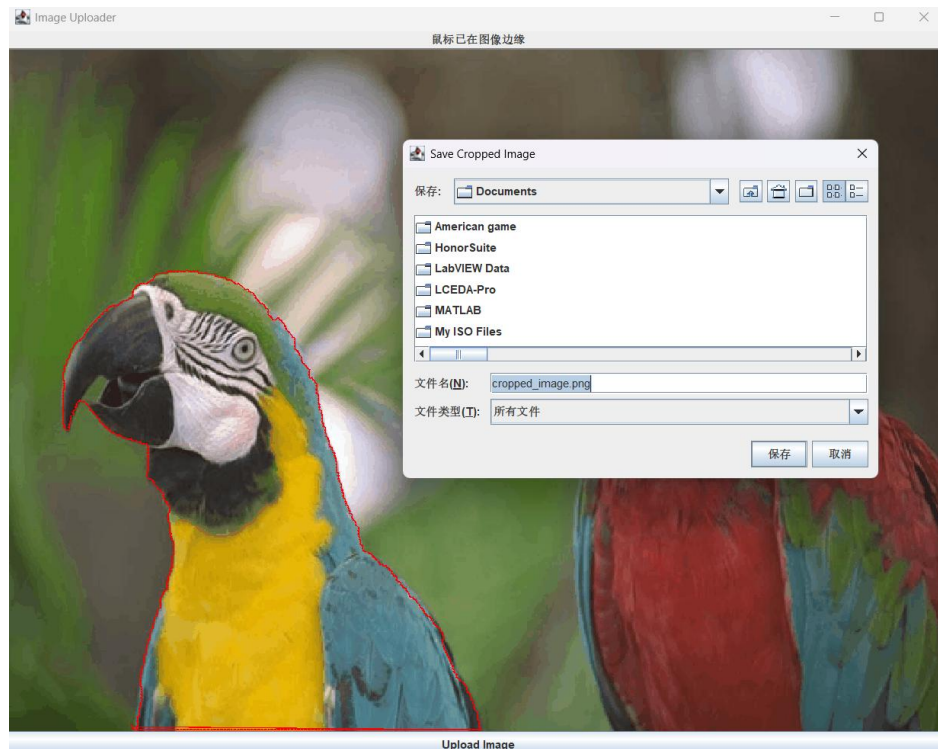
在弹出的这个界面中找到想要输入的图片， 将其打开

#### 5, 点击图片上想要截图的部分

截图前

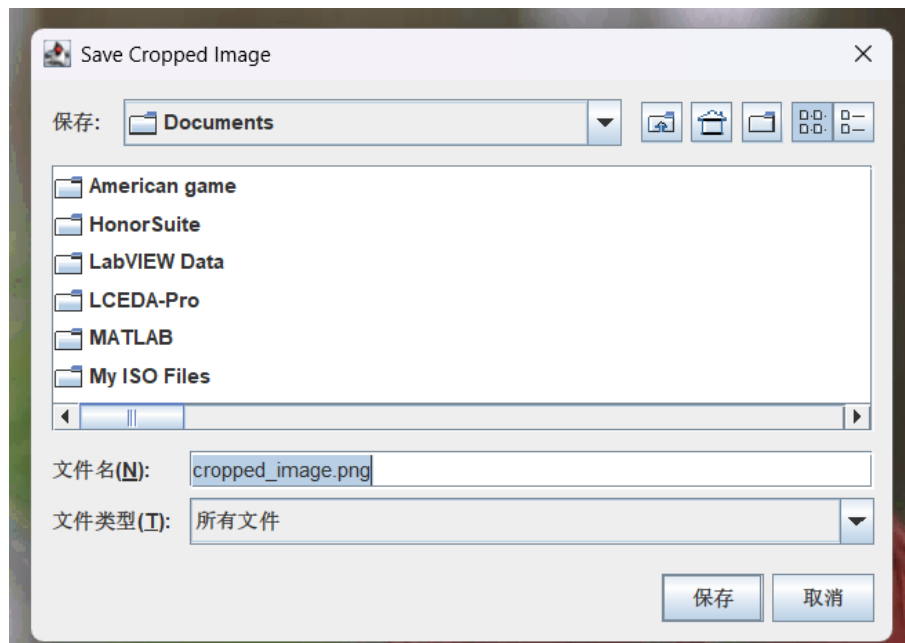


截图后

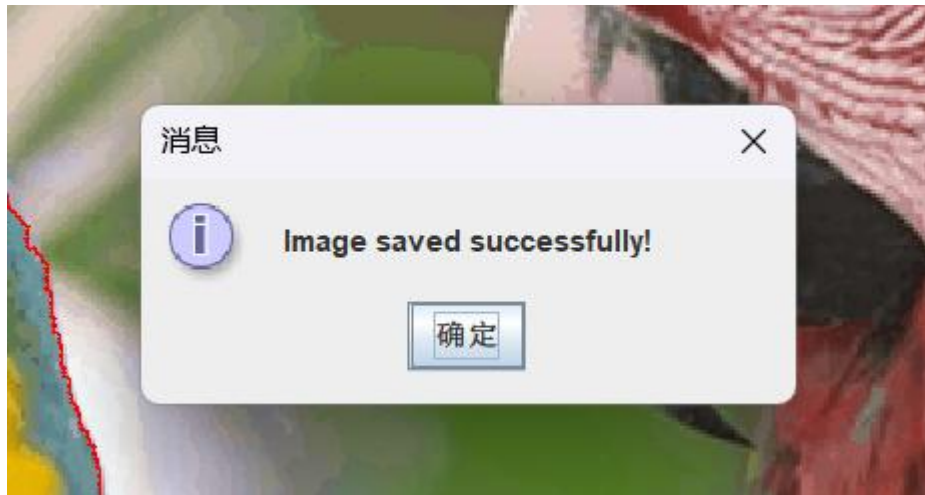


点击图片后，会开始记录截图路径（红色）。同时最上面的 label 每两秒会显示光标追踪的结果（鼠标已在图像边缘或鼠标已被自动移动到图像边缘）。同时，每五秒会变成进行了路径冷却。与老师的示例一样，光标回到起始点附近并点击才能完成截图并弹出保存截图的窗口。

## 6.选择存储截图路径



7，当选择完路径后，会弹出弹窗，显示截图成功



按下确定，会生成截图。

8，生成截图，并弹出窗口，窗口会显示你截的图。

