第二章 德塔JAVA 数据分析.

第一节 研发说明

德塔 Java 数据分析算法引擎系统说明书 V_1_0_2

作者: 罗瑶光
ID:430181198505250014
2019 年 8 月 8 日

1. 起源动机

2009 年报,作者在印度基督大学的数据结构实验室 与 Rohini 老师说,自己 能用 linklist 开发 大富翁游戏, 为了证明, 在书店花了 500 卢布买了本 Java how to program 6th 第一次接触 java.

2012 年作者在加州路德大学的计算机视觉课为了设计etl 节点处理像素流, 把作业设计成了API 包, 这是<德塔数据分析算法引擎系统>最早模型. 作者更新在 313699483 qq ID 中.

2013 年后的 ETL Unicorn 设计出来之后, 作者逐步设计 语音, 音像, 电影, 商业分析节点,于是将计算机视觉中的先贤们的算法 从 2 维 设计成 1 维 处理, 慢慢扩充优化丰富这个算法包.

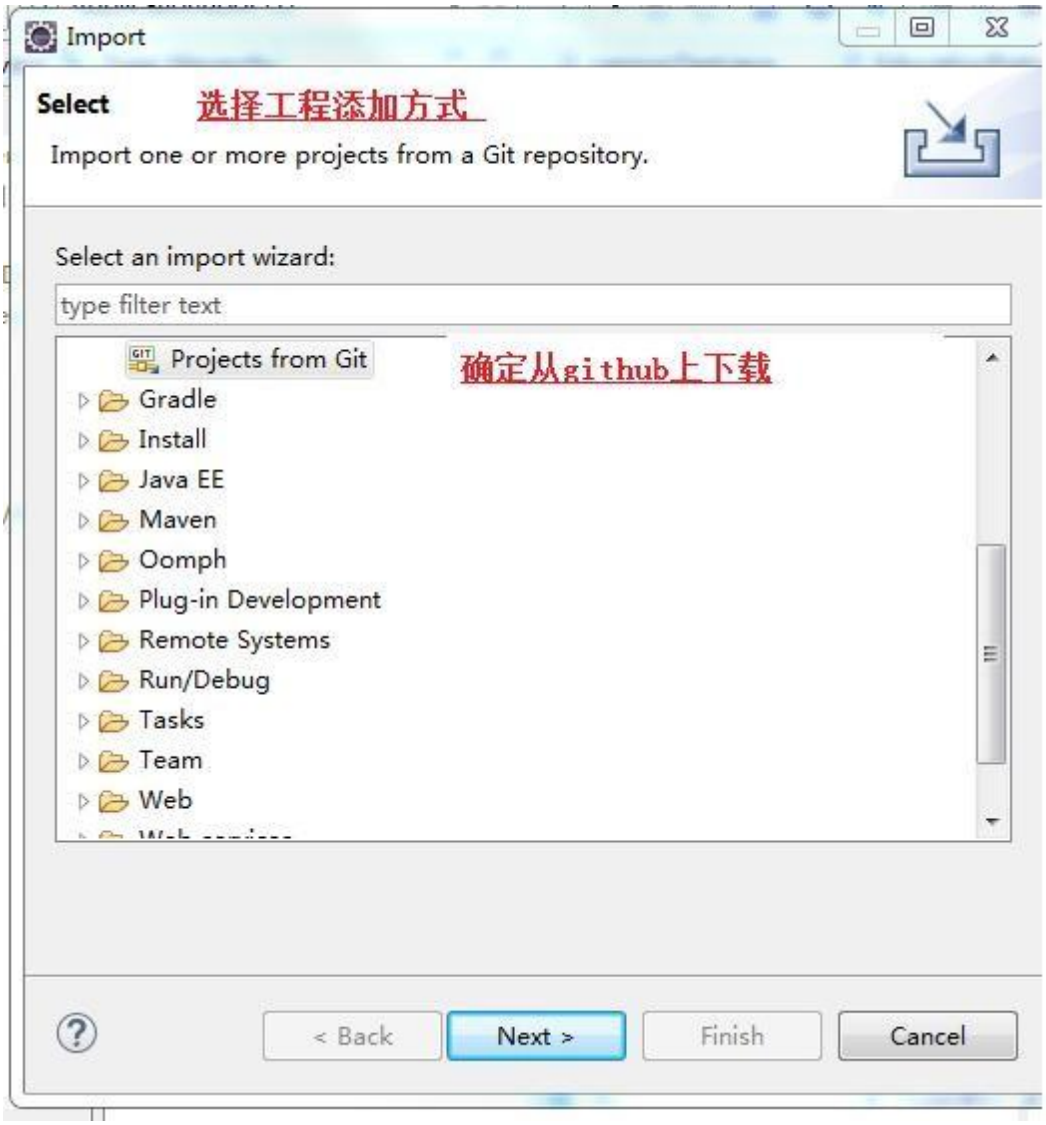
2. 应用特色:

适用于所有图片, 电影, 音乐的流数据处理. 作者一直在做算法加速优化.
适用于所有基于 波动粒子 工程的数据处理(天文, 地理, 勘探, 通讯, 盲分, 交通).

3. 使用方法

3.1 下载 java 开发软件:
Eclipse: https://www.eclipse.org/
Intellij:
https://www.jetbrains.com/idea/

3.2 导入 deta 图灵 api (API 是类库,接口 的意思, select 是选择 的意思)

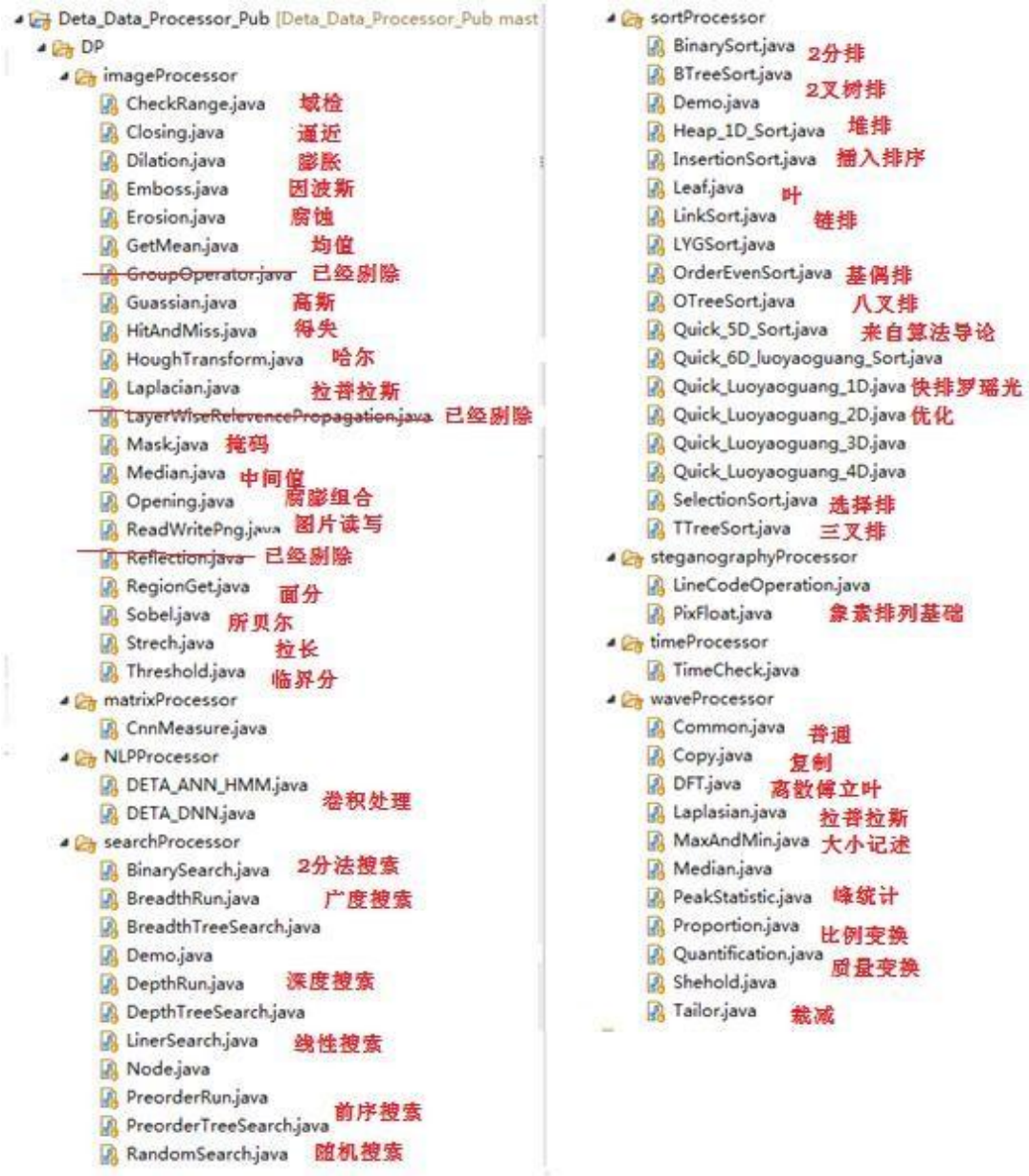


3.3 点 URI（uri 是互联网传输的一种协议规范关键字）

3.4 输入 Git 导入目标地址（git 是版本持续化控制软件，repository 是 git 工程的下载标识，host 是远程 主机，repository path 是git 工程 在主机上下载链接，protocol 是是通信协议，port 是端口，authentication 是密钥，user 是帐户名，password 是密码，store in secure store 是记录保存）

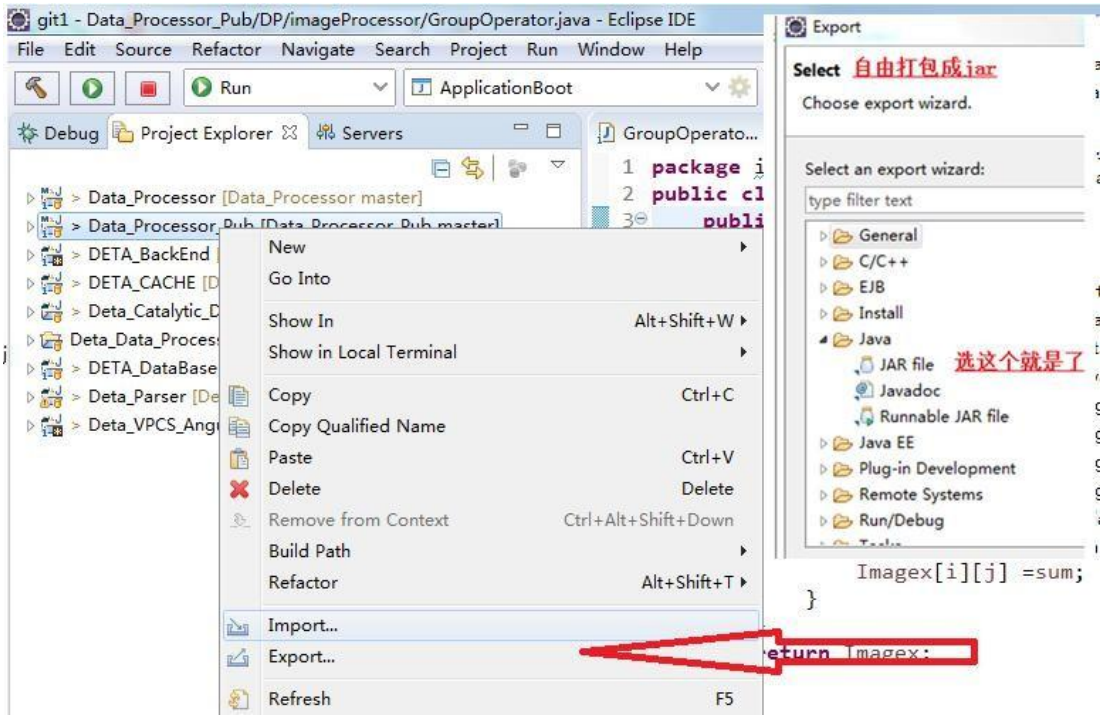

工程
https://github.com/yaoguangluo/Deta_Data_Processor_Pub.git
github.com
/yaoguangluo/Deta_Data_Processor_Pub.git
选 HTTPS443 协议
yaoguangluo

3.5 代码文件全局


Deta_Data_Processor_Pub [Deta_Data_Processor_Pub mast
DP
imageProcessor
CheckRange.java 域检
Closing.java 逼近
Dilation.java 膨胀
Emboss.java 因波斯
Erosion.java 腐蚀
GetMean.java 均值
GroupOperator.java 已经剔除
Guassian.java 高斯
HitAndMiss.java 得失
HoughTransform.java 哈尔
Laplacian.java 拉普拉斯
LayerWiseRelevencePropagation.java 已经剔除
Mask.java 掩码
Median.java 中间值
Opening.java 腐膨组合
ReadWritePng.java 图片读写
Reflection.java 已经剔除
RegionGet.java 面分
Sobel.java 所贝尔
Strech.java 拉长
Threshold.java 临界分
matrixProcessor
CnnMeasure.java
NLPProcessor
DETA_ANN_HMM.java 卷积处理
DETA_DNN.java
searchProcessor
BinarySearch.java 2分法搜索
BreadthRun.java 广度搜索
BreadthTreeSearch.java
Demo.java
DepthRun.java 深度搜索
DepthTreeSearch.java
LinerSearch.java 线性搜索
Node.java
PreorderRun.java
PreorderTreeSearch.java 前序搜索
RandomSearch.java 随机搜索
sortProcessor
BinarySort.java 2分排
BTreeSort.java 2叉树排
Demo.java
Heap_1D_Sort.java 堆排
InsertionSort.java 插入排序
Leaf.java 叶
LinkSort.java 链排
LYGSort.java
OrderEvenSort.java 基偶排
OTreeSort.java 八叉排
Quick_5D_Sort.java 来自算法导论
Quick_6D_luoyaoguang_Sort.java
Quick_Luoyaoguang_1D.java 快排罗瑶光
Quick_Luoyaoguang_2D.java 优化
Quick_Luoyaoguang_3D.java
Quick_Luoyaoguang_4D.java
SelectionSort.java 选择排
TTreeSort.java 三叉排
steganographyProcessor
LineCodeOperation.java
PixFloat.java 象素排列基础
timeProcessor
TimeCheck.java
waveProcessor
Common.java 普通
Copy.java 复制
DFT.java 离散傅立叶
Laplasian.java 拉普拉斯
MaxAndMin.java 大小记述
Median.java
PeakStatistic.java 峰统计
Proportion.java 比例变换
Quantification.java 质量变换
Shehold.java
Tailor.java 裁减

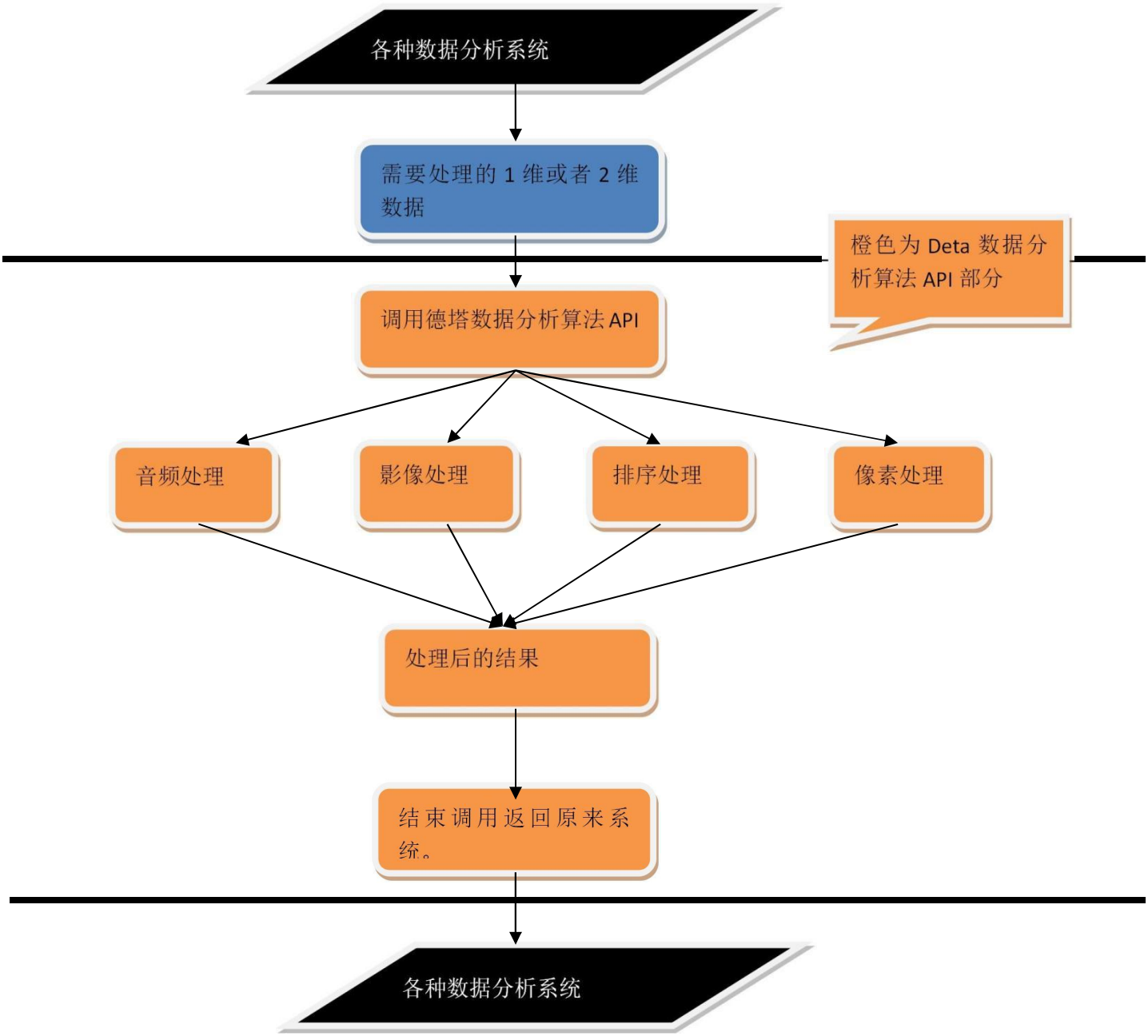
3.6 可下载的免费软件和图片例子集合：
https://github.com/yaoguangluo/Deta_Data_Processor_Pub
Github 非个软申请版本:_
https://github.com/yaoguangluo/Deta_Medicine_
https://github.com/yaoguangluo/Data_Processor
国内:
https://gitee.com/DetaChina/Data_Processor

3.8 可以任意 打包 jar 作为商业 库销售和集成.(jar 是 java 的库的意思, 可运行,可扩展, 可集成, export 是 打包输出的意思)



4. 功能注解：

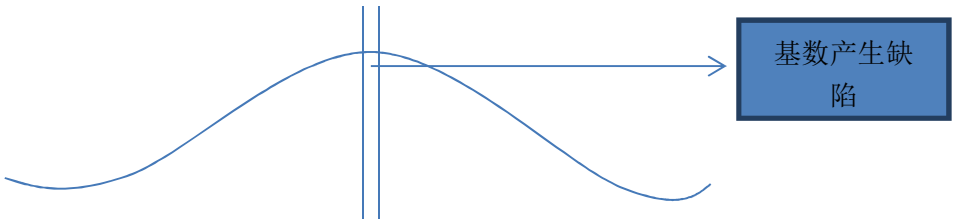
4.1 API 运行原理 Flowchat



4.2 罗瑶光小高峰过滤快排 4 代原理

快速排序的递归是基于 2 分拆分数组进行递归迭代的，假设数组的长为 n， 那么 2 分拆分将会有 2 种结果。当 n 为偶数时，则为完整拆分。当 n 为基数时，问题来了。n/2 将会有一定的概率 在 2 分递归的时候不对等。这个不对等产生的分配概率会形成许多短暂的运算小高峰时序。

概率的拆分如图：



如果中间拆分时为基数，将会产生概率分配误差的噪声峰，基于左右对比来确定一个线性方向，可以快速降解这个峰值。另外当一个单元的 while 循环次数增多时，峰值的利用增大，有效的计算性能增加。于是确定线性方向 x 选边缘最大数值。最后采用迪摩根分配律进行微分条件，性能再加速。于是比对算法快速排序有效的平滑这个问题： 如下：

```
private int partition(int[] a, int lp, int rp)
        { int x= a[lp]< a[rp]? a[lp]: a[rp];
        int lp1= lp;
        while(lp1< rp){
            while(!(a[lp1]>x|| lp1>=rp))
                { lp1++;
            }
            while(a[rp]>x){
                 rp--;
            }
            if(lp1<rp){
                int temp=a[rp];a[rp]=a[lp1];a[lp1]=temp;
            }
        }
        a[lp]=a[rp];a[rp]=x;
        return rp;
    }
```

两种比较领先的排序思维对比

罗瑶光手稿

今天我有阅读了一种这几年比较流行的快速排序思想，基于中轴分离进行左右分配的数列递归排序法。地址：github 上的 dongxingrong。 我仔细的研究了下，发现有几个地方可取，于是进行评价：

```
public int getMiddle(Integer[] list, int low, int high) {
        int tmp = list[low];    //数组的第一个作为中轴
        while (low < high) {
            while (low < high && list[high] >= tmp)
                { high--;
            }
            list[low] = list[high];     //比中轴小的记录移到低端
            while (low < high && list[low] <= tmp) {
                low++;
            }
            list[high] = list[low]; //比中轴大的记录移到高端
        }
        list[low] = tmp;            //中轴记录到尾
        return low;                 //返回中轴的位置
    }
```

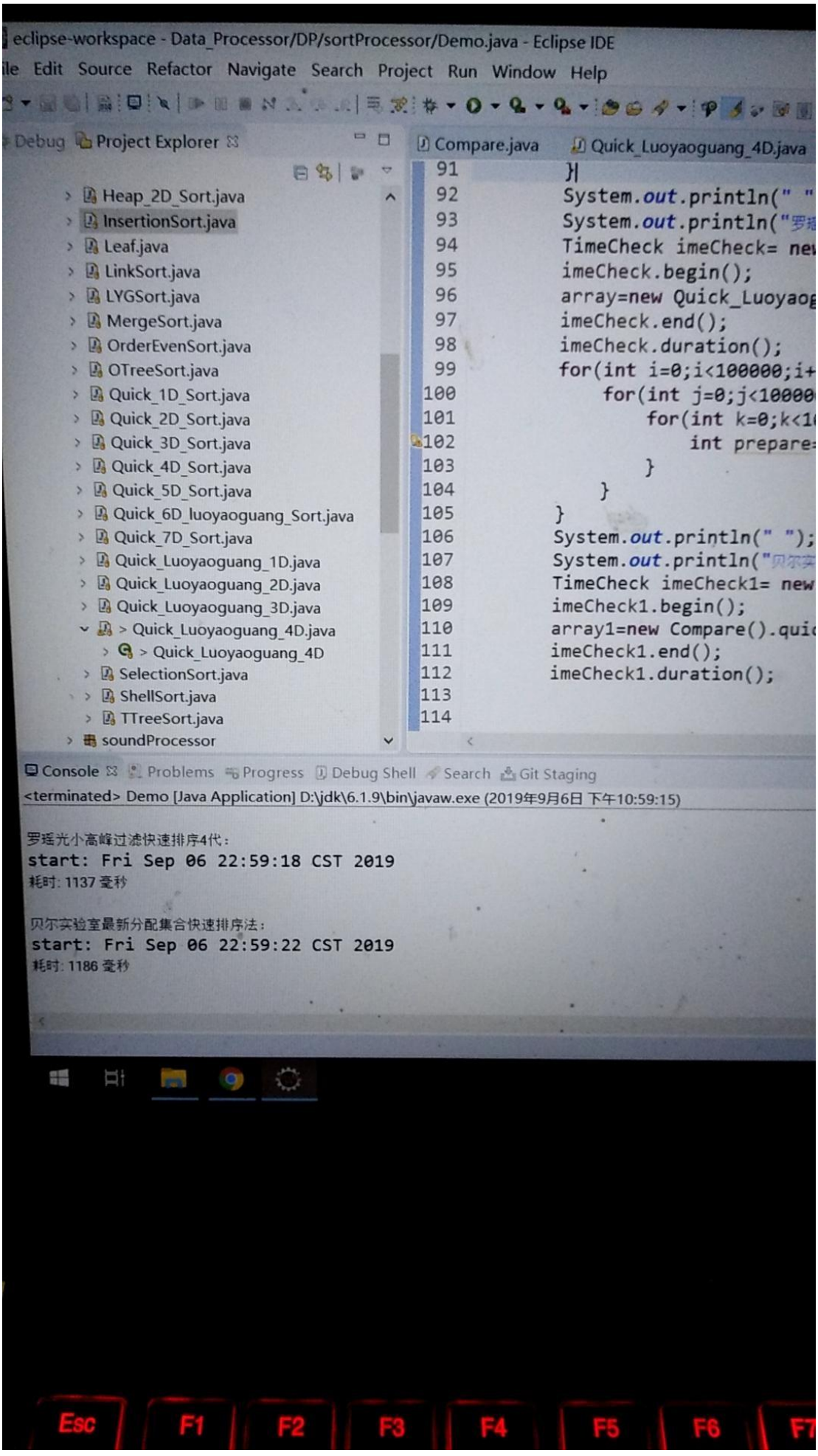
我思考了很久，在同频算子减少的方向中，直接减少 temp 的 swap 变量是一种先进的思想。但是利弊交错，我进行了和我的 4 代小高峰过滤快排做了详细对比：

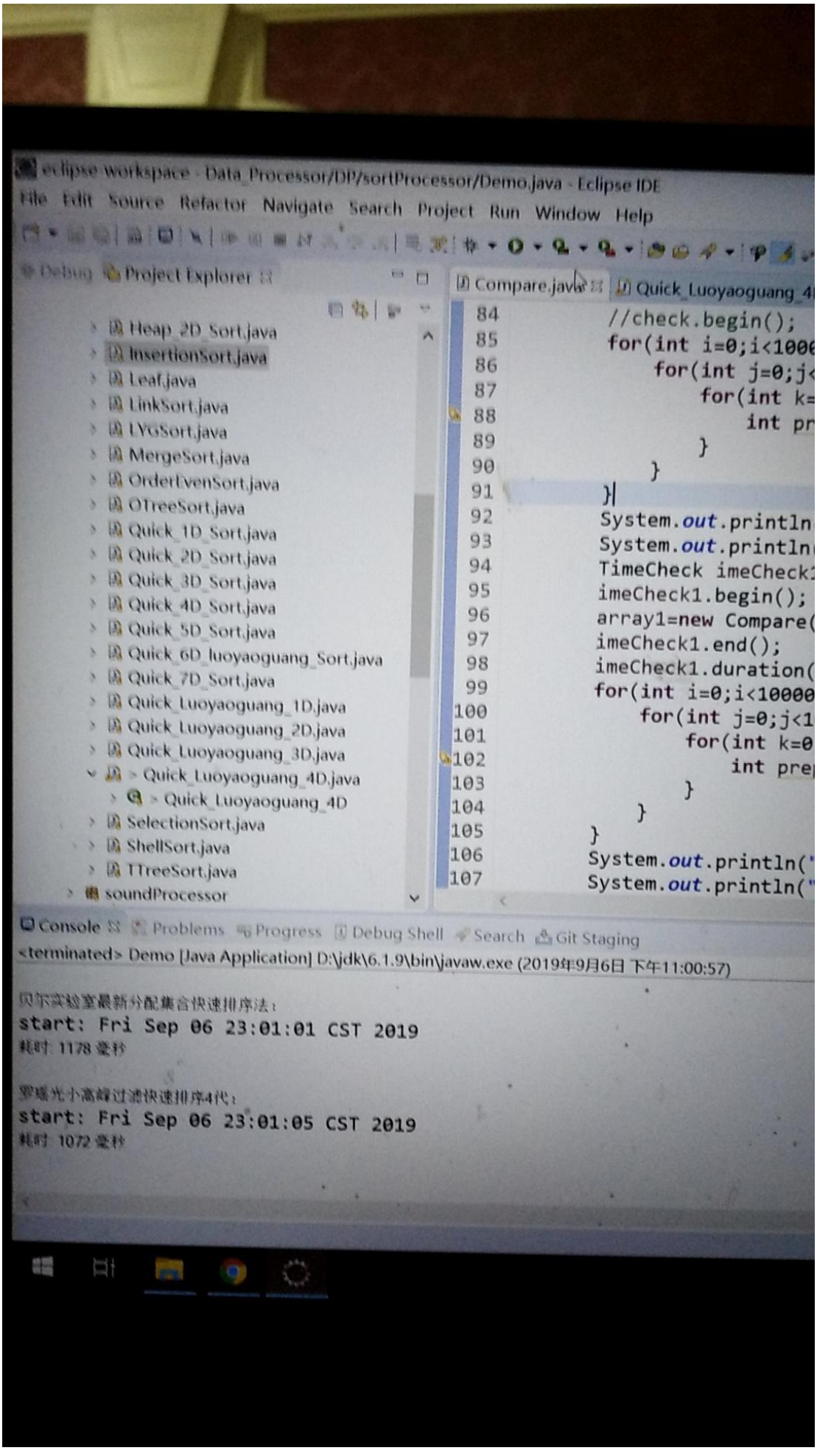
1000 万数列排序	瑶光高峰过滤	xingrong 中轴分配
算子减少	有	有
条件减少	有	有
离散效率	高	中
条件过滤	有	无
高频降解	无	有
计算性能	高	高
代码缩进	有	有
平均高峰先排耗时	1137 毫秒	1186 毫秒
平均分配先排耗时	1072 毫秒	1178 毫秒

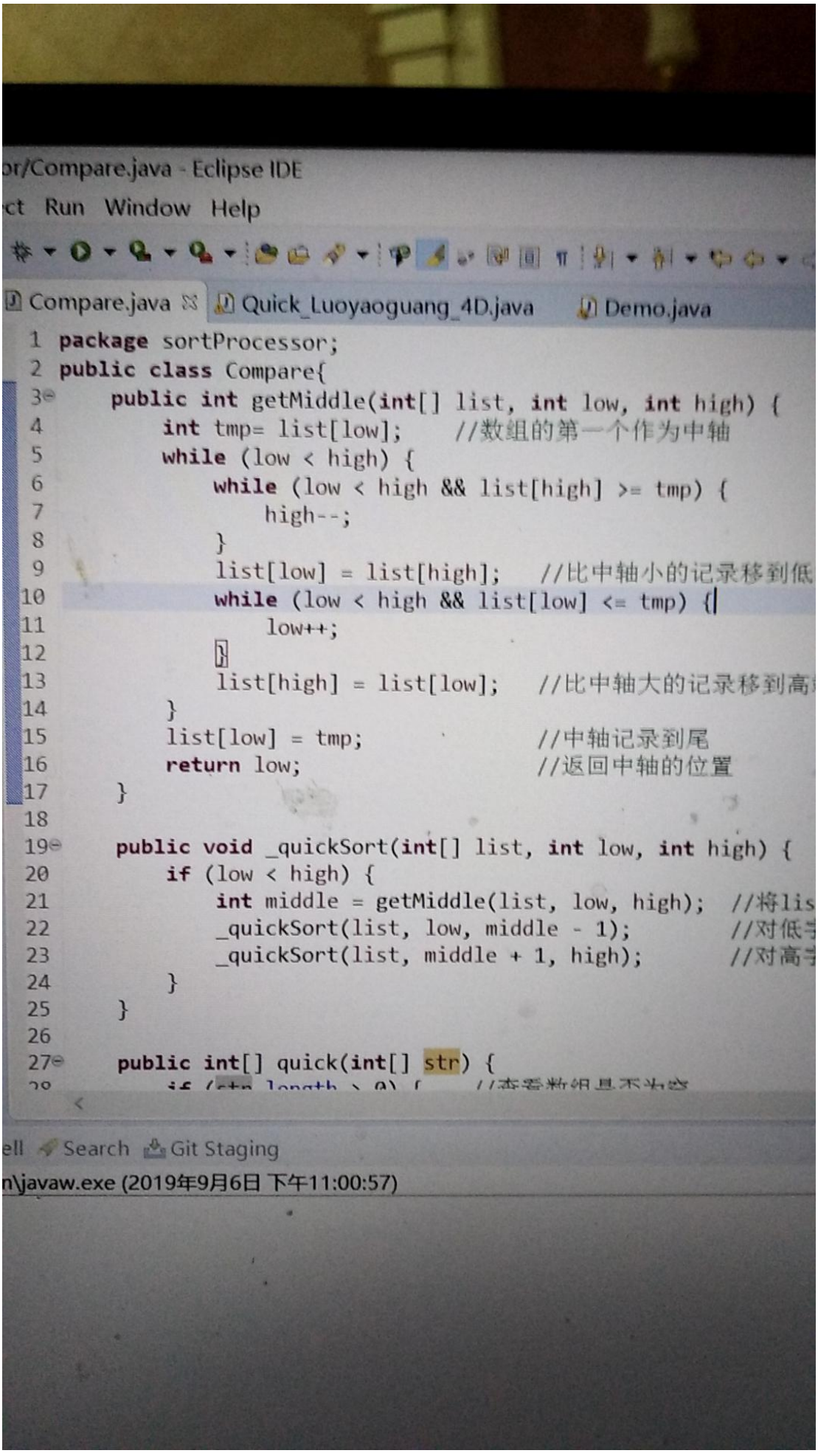
注解：这里的函数效率是指单个函数运行的计算消耗，包括堆栈消耗，内存阻塞，计算耗时等因素。 计算性能指的是数组的最大长度，单位消耗的时间，内存大小。

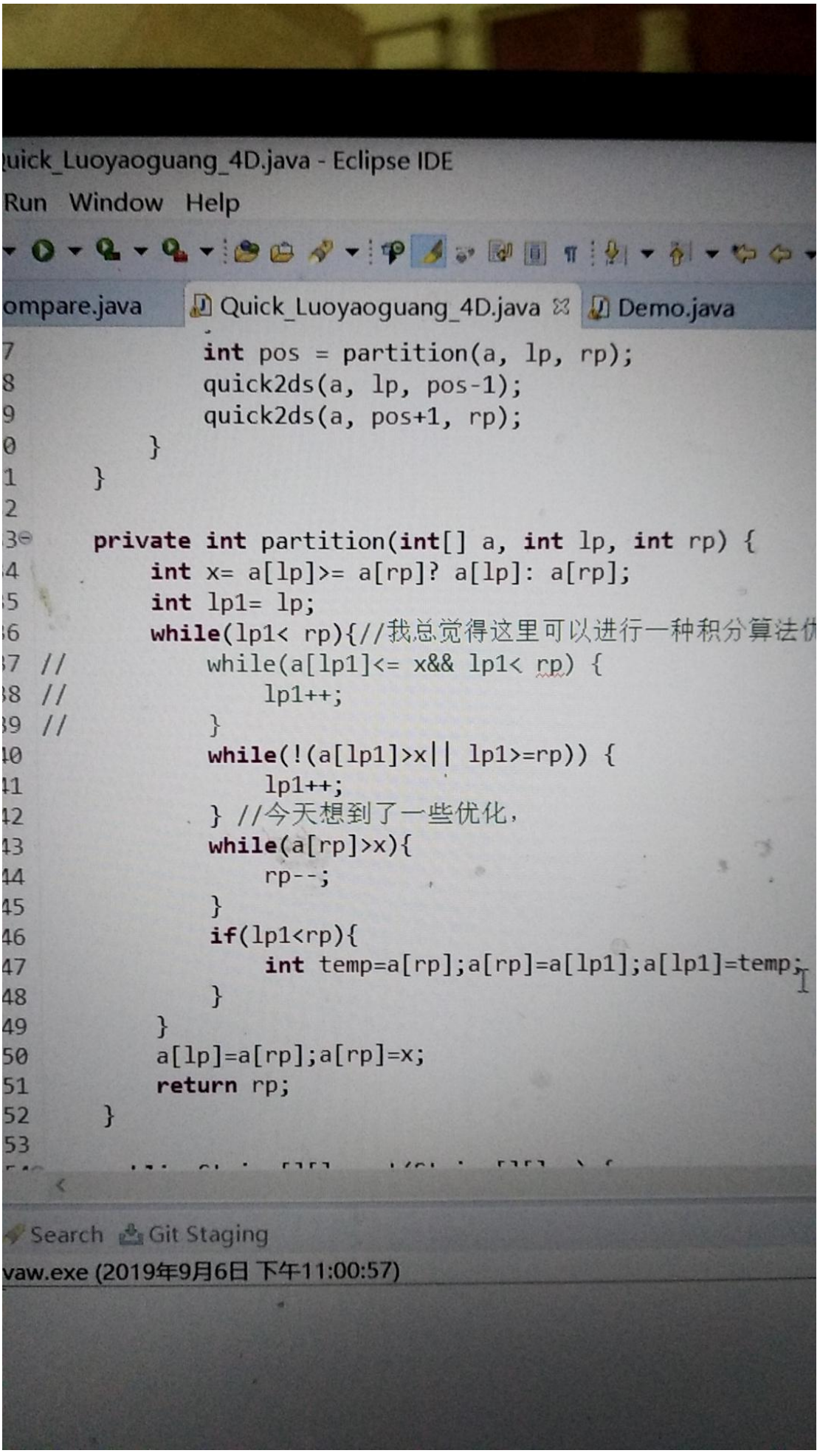
比较可以发现，因为递归的小高峰的存在，中轴心分配所需要的位移变换成为了必要执行功能，增大了开销， 这种开销可以最大限度的达到 balanced 递归逻辑的层数，但是对于指数方的多维算子条件缺少了过滤层，所以，性能耗损大。这种排序思想是完美的，为追求完美，丢失了一些东西，比较可惜。但是这种思想需要进行解析，在很多算法应用领域有宝贵的价值。

下面的 4 张图是堆 1000 万个随机数进行排序的比较耗时测试。


eclipse-workspace - Data_Processor/DP/sortProcessor/Demo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
Heap_2D_Sort.java
InsertionSort.java
Leaf.java
LinkSort.java
LYGSort.java
MergeSort.java
OrderEvenSort.java
OTreeSort.java
Quick_1D_Sort.java
Quick_2D_Sort.java
Quick_3D_Sort.java
Quick_4D_Sort.java
Quick_5D_Sort.java
Quick_6D_luoyaoguang_Sort.java
Quick_7D_Sort.java
Quick_Luoyaoguang_1D.java
Quick_Luoyaoguang_2D.java
Quick_Luoyaoguang_3D.java
> Quick_Luoyaoguang_4D.java
> Quick_Luoyaoguang_4D
SelectionSort.java
ShellSort.java
TTreeSort.java
soundProcessor
Compare.java
Quick_Luoyaoguang_4D.java
91 }
92 System.out.println(" "
93 System.out.println("罗
94 TimeCheck imeCheck= ne
95 imeCheck.begin();
96 array=new Quick_Luoyaog
97 imeCheck.end();
98 imeCheck.duration();
99 for(int i=0;i<100000;i+
100 for(int j=0;j<10000
101 for(int k=0;k<1
102 int prepare=
103 }
104 }
105 }
106 System.out.println(" ");
107 System.out.println("贝尔实
108 TimeCheck imeCheck1= new
109 imeCheck1.begin();
110 array1=new Compare().quic
111 imeCheck1.end();
112 imeCheck1.duration();
113
114
Console
Problems
Progress
Debug Shell
Search
Git Staging
<terminated> Demo [Java Application] D:\jdk\6.1.9\bin\javaw.exe (2019年9月6日 下午10:59:15)
罗瑶光小高峰过滤快速排序4代：
start: Fri Sep 06 22:59:18 CST 2019
耗时: 1137 毫秒
贝尔实验室最新分配集合快速排序法：
start: Fri Sep 06 22:59:22 CST 2019
耗时: 1186 毫秒
Esc
F1
F2
F3
F4
F5
F6
F7

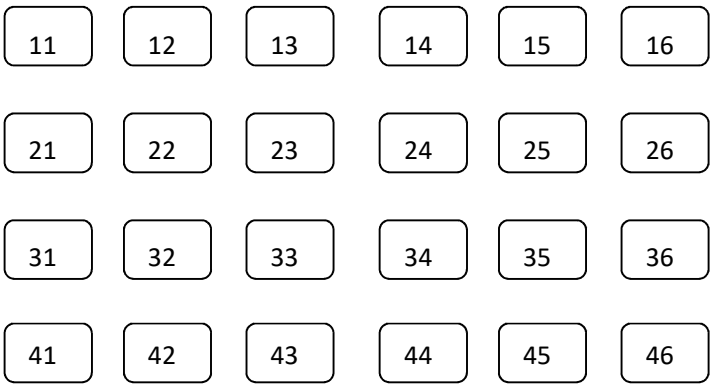

eclipse-workspace - Data_Processor/DP/sortProcessor/Demo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
Heap_2D_Sort.java
InsertionSort.java
Leaf.java
LinkSort.java
LYGSort.java
MergeSort.java
OrderEvenSort.java
OTreeSort.java
Quick_1D_Sort.java
Quick_2D_Sort.java
Quick_3D_Sort.java
Quick_4D_Sort.java
Quick_5D_Sort.java
Quick_6D_luoyaoguang_Sort.java
Quick_7D_Sort.java
Quick_Luoyaoguang_1D.java
Quick_Luoyaoguang_2D.java
Quick_Luoyaoguang_3D.java
Quick_Luoyaoguang_4D.java
Quick_Luoyaoguang_4D
SelectionSort.java
ShellSort.java
TTreeSort.java
soundProcessor
Compare.java
Quick_Luoyaoguang_4
84 //check.begin();
85 for(int i=0;i<1000
86 for(int j=0;j<
87 for(int k=
88 int pr
89 }
90 }
91 }
92 System.out.println
93 System.out.println
94 TimeCheck imeCheck
95 imeCheck1.begin();
96 array1=new Compare(
97 imeCheck1.end();
98 imeCheck1.duration(
99 for(int i=0;i<10000
100 for(int j=0;j<1
101 for(int k=0
102 int pre
103 }
104 }
105 }
106 System.out.println(
107 System.out.println(
Console
Problems
Progress
Debug Shell
Search
Git Staging
<terminated> Demo [Java Application] D:\jdk\6.1.9\bin\javaw.exe (2019年9月6日 下午11:00:57)
贝尔实验室最新分配集合快速排序法：
start: Fri Sep 06 23:01:01 CST 2019
耗时 1178 毫秒
罗耀光小高峰过滤快速排序4代：
start: Fri Sep 06 23:01:05 CST 2019
耗时 1072 毫秒


or/Compare.java - Eclipse IDE
ct Run Window Help
Compare.java
Quick_Luoyaoguang_4D.java
Demo.java
1 package sortProcessor;
2 public class Compare{
3 public int getMiddle(int[] list, int low, int high) {
4 int tmp= list[low]; //数组的第一个作为中轴
5 while (low < high) {
6 while (low < high && list[high] >= tmp) {
7 high--;
8 }
9 list[low] = list[high]; //比中轴小的记录移到低
10 while (low < high && list[low] <= tmp) {
11 low++;
12 }
13 list[high] = list[low]; //比中轴大的记录移到高
14 }
15 list[low] = tmp; //中轴记录到尾
16 return low; //返回中轴的位置
17 }
18
19 public void _quickSort(int[] list, int low, int high) {
20 if (low < high) {
21 int middle = getMiddle(list, low, high); //将lis
22 _quickSort(list, low, middle - 1); //对低
23 _quickSort(list, middle + 1, high); //对高
24 }
25 }
26
27 public int[] quick(int[] str) {
ell Search Git Staging
n\javaw.exe (2019年9月6日 下午11:00:57)


uick_Luoyaoguang_4D.java - Eclipse IDE
Run Window Help
ompare.java
Quick_Luoyaoguang_4D.java
Demo.java
int pos = partition(a, lp, rp);
quick2ds(a, lp, pos-1);
quick2ds(a, pos+1, rp);
}
}
private int partition(int[] a, int lp, int rp) {
int x= a[lp]>= a[rp]? a[lp]: a[rp];
int lp1= lp;
while(lp1< rp){//我总觉得这里可以进行一种积分算法优
// while(a[lp1]<= x&& lp1< rp) {
// lp1++;
// }
while(!(a[lp1]>x|| lp1>=rp)) {
lp1++;
} //今天想到了一些优化，
while(a[rp]>x){
rp--;
}
if(lp1<rp){
int temp=a[rp];a[rp]=a[lp1];a[lp1]=temp;
}
}
a[lp]=a[rp];a[rp]=x;
return rp;
}
Search
Git Staging
vaw.exe (2019年9月6日 下午11:00:57)

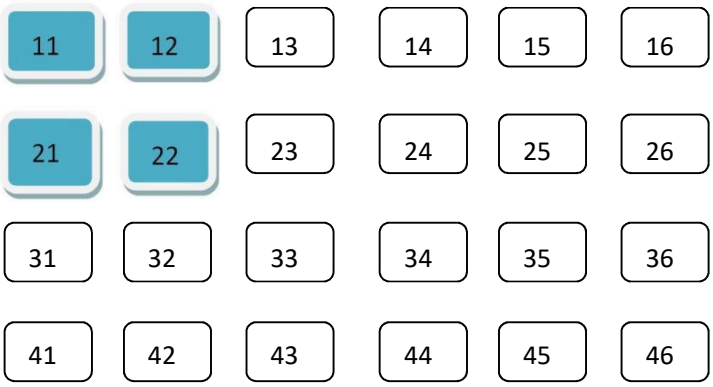
4.3 维度卷积计算原理：

4.3.1 给定一个需要处理的矩阵如下：



4.3.2 给一个卷积核算子如下：

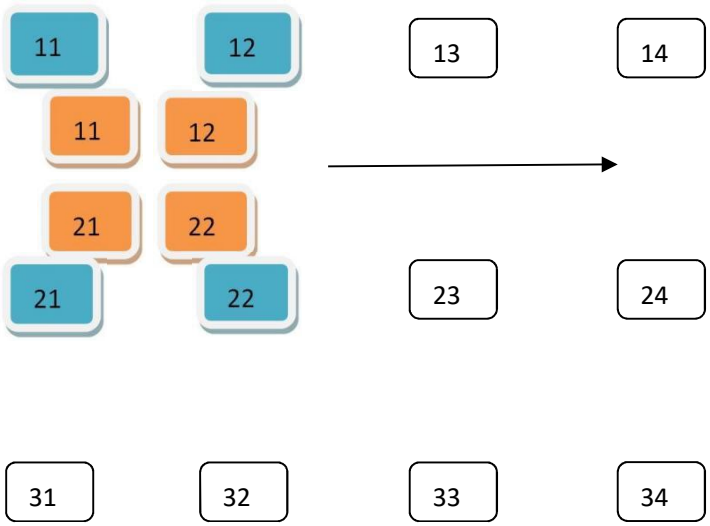
4.3.3 卷积处理方式：通过蓝色部分和橙色部分进行相应的内核计算操作，进行局部卷积微积分处理数据矩阵。



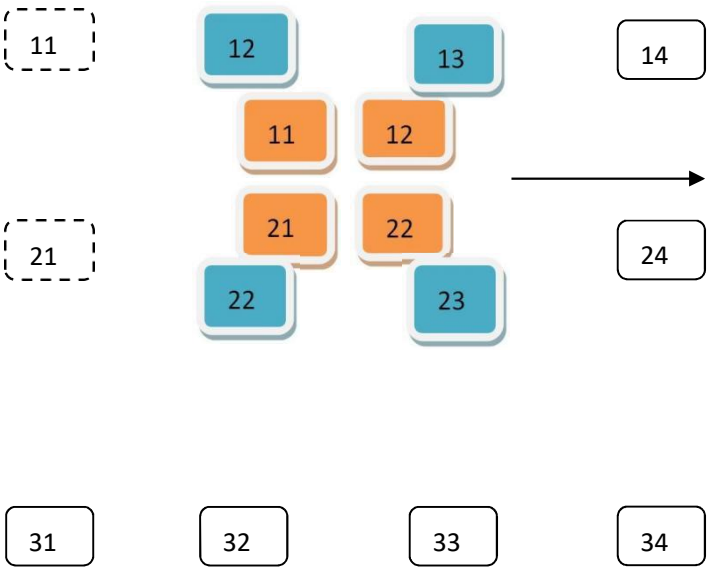
注解：我再这里给出例子是 长为 2 宽为 2 的内核，根据实际情况可以自定义修改，如果像处理矩阵边缘数据，需要反射扩张增加原矩阵的长和宽。

4.3.4 先做红色和蓝色的部分卷积 相同颜色做相应算法运算如下

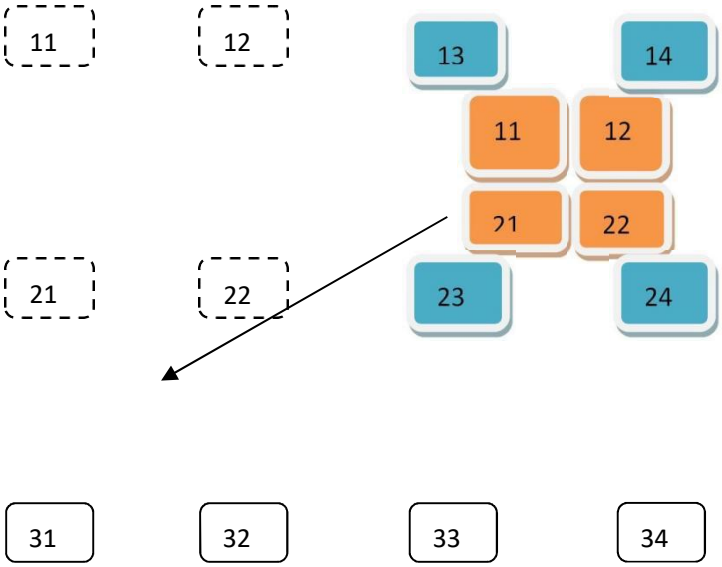
4.3.5 从左到右进行相应
卷积运算



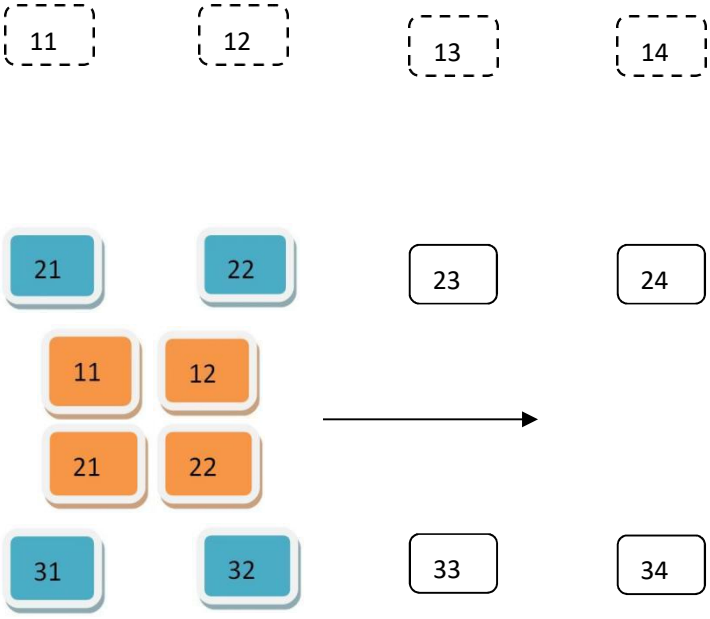
4.3.6 从左到右进行相应卷积运算，运算的虚线集为已经处理过的待输出数据。



4.3.7 到了右边界 宏诺伊曼的思想从上到下处理。



4.3.8 然后再从左到右相应卷积处理。



注解：卷积算子有很多种，根据不同的工程应用，请选择合适的算法卷积算子

5. 适用范围

适用于所有图片，电影，音乐的流数据处理. 作者一直在做算法加速优化.
适用于所有基于 波动粒子 工程的数据处理(天文，地理，勘探，通讯，盲分，交通).

6. 注意

注意 1: 该作品免费版本使用权由国际软件研发协议 apache-2.0 证书保护. 任何单位任意修改集成使用时请标注关键字:“罗瑶光”

注意 2 本系统 基于先贤的算法理论思想进行 Java 编译，排名不分先后.
先贤名单: 祖冲之，高斯，傅立叶，牛顿，薛定锷，拉普拉斯，巴特沃斯，所贝尔，哈尔，因波斯，迪摩根，霍夫 等.
注意 3 本系统 中的先贤算法思想 作者 无权所有，有权应用. 同样客户和使用者 可以自由化应用该 软件系统，但无权所有其 Refer 的先贤算法 以及思想理论的国籍归属等一系列所属权修改.

注意 4: 当前版本是 1.0.0，一直在优化中,有任何 bug 请直接联系作者.

QQ: 2080315360 (qq: 腾讯)
WECHAT: 15116110525 (WECHAT 微信)
TEL: 15116110525 (tel: 电话号码)
EMAIL: 2080315360@qq.com (email: 邮件地址)

6.1 德塔数据分析实现集合涉及的先贤理论思想和发现。互联网链接：

傅里叶思想：
https://www.doc88.com/p-0973999656458.html
https://baike.sogou.com/v102060438.htm?fromTitle=%E5%82%85%E9%87%8C%E5%8F%B6

拉普拉斯思想：
https://baike.sogou.com/v815845.htm?fromTitle=%E6%8B%89%E6%99%AE%E6%8B%89%E6%96%AF%E7%AE%97%E5%AD%90
https://baike.sogou.com/v181283.htm?fromTitle=%E6%8B%89%E6%99%AE%E6%8B%89%E6%96%AF

浮雕思想：
https://en.wikipedia.org/wiki/Image_embossing

高斯思想：
https://baike.sogou.com/v7811633.htm?fromTitle=%E9%AB%98%E6%96%AF%E6%A0%B8%E5%87%BD%E6%95%B0

腐蚀膨胀思想：
https://baike.sogou.com/v178215318.htm?fromTitle=%E9%97%AD%E8%BF%90%E7%AE%97

索贝尔思想：
https://baike.sogou.com/v101294427.htm?fromTitle=Sobel%E7%AE%97%E5%AD%90

Cnn：
https://en.wikipedia.org/wiki/Convolutional_neural_network

Rnn：
https://en.wikipedia.org/wiki/Recursive_neural_network

Ann：
https://en.wikipedia.org/wiki/Artificial_neural_network

Lwa:

https://baike.sogou.com/v11009683.htm?fromTitle=bp%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C
https://baike.sogou.com/v11009683.htm?fromTitle=bp 神经网络

BST:
https://baike.sogou.com/v71134459.htm?fromTitle=AVL%E6%A0%91

Quicksort:
https://baike.sogou.com/v7638866.htm?fromTitle=%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F

Heapsort:
https://baike.sogou.com/v4838322.htm?fromTitle=%E5%A0%86%E7%A7%AF%E6%8E%92%E5%BA%8F

Selectsort:
https://baike.sogou.com/v73874629.htm?fromTitle=%E9%80%89%E6%8B%A9%E7%B1%BB%E6%8E%92%E5%BA%8F%E6%B3%95

Insertsort:
https://baike.sogou.com/v7832934.htm?fromTitle=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F

7. 感谢

1 感谢 Renhat 教授 提供了 Reflection 函数(已经移出个人著作权范围)
2 感谢 同学 罗阳参与设计 group 函数(已经移出个人著作权范围)
3 感谢 同学 高顺参与讨论 Hull 算子不稳定熵(已经移出个人著作权范围)
4 感谢 Vicent Boucher 提供的LWA 理论思想(已经移出个人著作权范围)
5 感谢 同学 Mahesh 参与讨论 linklist 的C 语言快速书写方式(已经移出个人著作权范围)
Deta 项目设计 采用 Mind Master 软件.
Deta 项目研发 采用 Eclipse IDE 软件.
Deta 项目测试 采用 JUNIT API 软件.
Deta 项目作品 主要采用 JAVA JDK8+.
Deta 项目编码和算法基础能力来自作者在印度基督大学 学习的 数据结构 课程. 作者长期使用 微星 windows 7 操作系统开发此项目, 电脑装 360 杀毒软件保证其高效研发环境. 感谢 github 和gitee 备份, 节省了作者 大量的存储硬盘, 同时方便 查阅, 逻辑 的鼠标键盘 给作者 提供了迅捷 的输入输出 便利 .当然 电信的网络, 老爸,老妈, 都要感谢的.

8 研发需要清单

8.1 Java 编辑器.

8.2 Jdk8+. Java 虚拟机运行环境.

8.4 一台连网的电脑.

8.3 Junit 测试包.

研发源码

#函数名：像素处理包像素边际检查
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：这个函数用于像素存储在矩阵中超出0~255的正常取值空间进行自动边值过滤矫正

```
package imageProcessor;
import java.io.IOException;
public class CheckRange {
    public int[][] Processor(int[][] g) throws IOException
        { for(int i=0;i<g.length;i++){
            for(int
                j=0;j<g[0].length;j++){ if(
                g[i][j]<0){
                    g[i][j]=0;
                }
                if (g[i][j]>255){
                    g[i][j]=255;
                }
            }
        }
        return g;
    }
}
```

==========

#函数名：像素细化膨胀处理中的逼近连接处理
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：像素在细化过程中的膨胀逼近连接处理。

```
package imageProcessor;
public class Closing {
    public int[][] Processor(int[][]g,int[][]kenel){
        int[][] closing= new Dilation().Processor(g, kenel);
        closing= new Erosion().Processor(closing,kenel);
        return closing;
    }
}
```

==========

#函数名：像素细化膨胀处理中的逼近处理核心算法
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：像素在细化过程中的膨胀逼近处理核心算法。
#函数解析：同过一个0和1的掩码矩阵来进行2维循环保留有效像素数据。Flection用于矩阵的四周复制内核的一半大小 #避免溢出数组最大值和最小值。

```
package imageProcessor;
import imageProcessor.Reflection;
public class Dilation{
    int[][]kernel;
    int[][]GetRegion;
```

```
    int [][]tempG;
    private void initTemp(int[][]thisG) {
        tempG=new int[thisG.length][thisG[0].length];
        for(int i=0;i<thisG.length;i++){
            for(int
                j=0;j<thisG[0].length;j++){ tem
                pG[i][j]=0;
            }
        }
    }
    public int[][] Processor(int[][] g, int[][] kenel)
        { int thisw=g.length;
        int thish=g[0].length;
        int [][]thisG = null;
        if((kenel.length%2==0)&&(kenel[0].length%2==0))
            thisG= new Reflection().PadImage(g, kenel.length+1, kenel[0].length+1);
        if((kenel.length%2==0)&&(kenel[0].length%2==1))
            thisG= new Reflection().PadImage(g, kenel.length+1, kenel[0].length);
        if((kenel.length%2==1)&&(kenel[0].length%2==0))
            thisG= new Reflection().PadImage(g, kenel.length, kenel[0].length+1);
        if((kenel.length%2==1)&&(kenel[0].length%2==1))
            thisG= new Reflection().PadImage(g, kenel.length, kenel[0].length);
        initTemp(thisG);
        int w=kenel.length;
        int h=kenel[0].length;
        for(int
            i=w/2;i<thisw+w/2;i++){ for(int
            j=h/2;j<thish+h/2;j++){
                for(int
                    p=0;p<w;p++){ for(int
                    q=0;q<h;q++){
                        if(kenel[p][q]==1){
                            if(thisG[i][j]>0){
                                tempG[i-w/2+p][j-h/2+q]=thisG[i][j];
                            }
                        }
                    }
                }
            }
        }
        return tempG;
    }
}
```

==========

#函数名：浮雕卷积算法

#函数思想作者：无法找到

#函数功能作者：罗瑶光

#函数用途：用于像素的类似浮雕化0~255分布变换。
#函数解析：通过对角极值的内核卷积对像素矩阵进行刻度过滤变换，形成浮雕效果。

```
package imageProcessor;
import java.io.IOException;
public class Emboss {
    public int[][] Processor( int[][] g) throws IOException
        { int[][] refG =new Reflection().PadImage(g, 3, 3);
        int[]size = {g.length,g[0].length};
        double[][] Gx = new double [size[0]][size[1]];
        int[][] gxk = new int[][]{
            {-1, 0, 0},
            { 0, 0, 0},
            { 0, 0, 1}};
            //GROUP OPERATION
            Gx= new GroupOperator().GO(gxk, refG, size);
            for(int i=0;i<Gx.length;i++){
                for(int
                    j=0;j<Gx[0].length;j++){ g[i][j]
                    =(int) (Gx[i][j]/2+127);
                }
            }
            int out[][] = new int[g.length][g[0].length];
            for (int i = 0; i < g.length; i++) {
                for (int j = 0; j < g[0].length; j++)
                    { out[i][j]= g[i][j];
                }
            }
            return  new CheckRange().Processor(out);
    }
}
```

==========

#函数名：腐蚀卷积算法
#函数思想作者：无法找到
#函数功能作者：罗瑶光
#函数用途：用于像素的面机分布进行边缘瘦化缩小。
#函数解析：通过掩码矩阵的形态进行有规律的自适应边缘瘦化。

```
package imageProcessor;
import imageProcessor.Reflection;
public class Erosion{
    int[][]kernel;
    int[][]GetRegion;
    int [][]tempG;
    private void initTemp(int[][]thisG) {
        tempG=new int[thisG.length][thisG[0].length];
        for(int i=0;i<thisG.length;i++){
            for(int
                j=0;j<thisG[0].length;j++){ tem
```

```
                pG[i][j]=thisG[i][j];
            }
        }
    }
    public int[][] Processor(int[][] g, int[][] kenel)
        { int thisw=g.length;
        int thish=g[0].length;
        int [][]thisG = null;
        if((kenel.length%2==0)&&(kenel[0].length%2==0))
            thisG= new Reflection().PadImage(g, kenel.length+1, kenel[0].length+1);
        if((kenel.length%2==0)&&(kenel[0].length%2==1))
            thisG= new Reflection().PadImage(g, kenel.length+1, kenel[0].length);
        if((kenel.length%2==1)&&(kenel[0].length%2==0))
            thisG= new Reflection().PadImage(g, kenel.length, kenel[0].length+1);
        if((kenel.length%2==1)&&(kenel[0].length%2==1))
            thisG= new Reflection().PadImage(g, kenel.length, kenel[0].length);
        initTemp(thisG);
        int w=kenel.length;
        int h=kenel[0].length;
        for(int
            i=w/2;i<thisw+w/2;i++){ for(int
            j=h/2;j<thish+h/2;j++){
                for(int
                    p=0;p<w;p++){ for(int
                    q=0;q<h;q++){
                        if(kenel[p][q]==1){
                            if(thisG[i][j]==0){
                                tempG[i-w/2+p][j-h/2+q]=0;
                            }

                        }
                    }
                }
            }
        }
        return tempG;
    }
}
```

==========

#函数名：获取平均值算法

#函数思想作者：无

#函数功能作者：罗瑶光

#函数用途：用于得到一个像素矩阵独有的不变平均值

#函数解析：这个平均值是个不变值。用于像素矩阵的一些快速识别，均值卷积等基础功能，用途很多。

```
package imageProcessor;
public class GetMean {
    public double print_Mean(int[][] outDIR)
```

```
        { double sum = 0;
        double meanValue = sum;
        for(int
            i=0;i<outDIR.length;i++){ for(int
            j=0;j<outDIR[0].length;j++){
                sum += outDIR[i][j];
            }
        }
        meanValue = sum/( outDIR.length * outDIR[0].length );
        return meanValue;
    }
}
```

==========

#函数名：高斯2维卷积过滤算法

#函数思想作者：数学家：高斯

#函数功能作者：罗瑶光

#函数用途：用于对像素矩阵邻接噪进行平滑过滤。

#函数解析：通过高斯数学家发明的邻接数据平滑变换算法生成的内核进行带有精度的卷积平滑处理。

```
package imageProcessor;
import java.awt.image.BufferedImage;
import java.io.IOException;
public class Guassian {
    public int[][] Processor2D(int[][] g,int d,int e,double sig) throws IOException
        { int [][]flac_grn=new Reflection().PadImage(g, d, e);
        double kenel[][] = new double[d][e];
        //GAUSSIAN KENEL
        double sumhere = 0;
        double t = 0;
        for(int k = 0; k<d;
            ++k){ for(int 1 = 0; 1<e;
            ++1){
//高斯核心公式开始
                t = (Math.pow(k-(d/2), 2)+Math.pow(1-(e/2), 2))/(2*Math.pow(sig, 2));
                t = Math.exp(-t);
                t = (1* t)/(2*Math.PI*Math.pow(sig, 2));
//高斯核心公式结束
                kenel[k][1] = t;
                sumhere = sumhere + kenel[k][1];
            }
        }
        System.out.println("--->"+sumhere);
        //--normalization
        double sum1=0;
        for(int i=0; i<d;
            ++i){ for(int j=0; j<e;
            ++j){
                kenel[i][j] = kenel[i][j]/sumhere;
```

```
                sum1 = sum1 + kenel[i][j];
            }
        }
        //--end of producing gaussian matrix
        System.out.println("gaussian sum: " + sum1);
        //GAUSSIAN pro
        //int[][] neib=new int[15][15];//for sort
        for (int i=d/2;i<g.length+d/2;i++){
            for(int
                j=e/2;j<g[0].length+e/2;j++){ dou
                ble sum=0;
                for(int
                    k=0;k<d;k++){ for(int
                    l=0;l<e;l++){
                        sum=(float)(sum+flac_grn[i+k-d/2][j+l-e/2]
                                * kenel[k][l]);
                    }
                }
                // System.out.println(sum);
                g[i-d/2][j-e/2]=(int)(sum);
            }
        }
        return new CheckRange().Processor(g);
    }
    public BufferedImage Processor(BufferedImage lygimage, double d, double e
            ,double k) throws IOException {
        int r[][]=new ReadWritePng().REDpngRead(lygimage);
        int g[][]=new ReadWritePng().GRNpngRead(lygimage);
        int b[][]=new ReadWritePng().BLUpngRead(lygimage);
        //r[][]
        r=Processor2D(r,(int)d,(int)e,k);
        //g[][]
        g=Processor2D(g,(int)d,(int)e,k);
        //b[][]
        b=Processor2D(b,(int)d,(int)e,k);
        //r[][]g[][]b[][] to image
        lygimage=new ReadWritePng().createBufferImage(r,g,b);
        return lygimage;
    }
}
```

==========

#函数名：丢失与得到冗余算法

#函数思想作者：无法找到

#函数功能作者：罗瑶光

#函数用途：用于处理带有环形的像素分布的深度腐蚀过滤变换

#函数解析：用于处理图片的凹点的标注。

```
package imageProcessor;
```

```
import java.io.IOException;
import imageProcessor.Reflection;
public class HitAndMiss{
    int[][] kernel;
    int[][] GetRegion;
    int[][] tempG;
    private void initTemp(int[][]thisG) {
        tempG= new int[thisG.length][thisG[0].length];
        for(int i= 0; i< thisG.length; i++){
            for(int j= 0; j< thisG[0].length;
                j++){ tempG[i][j]= 0;
            }
        }
    }
    public int[][] Processor(int[][] g, int pix, int[][] kenel) throws
        IOException{ int [][]g1= new int[g.length][g[0].length];
        for(int i=0; i<g.length;i++){
            for(int j=0; j<g[0].length;
                j++){ g1[i][j]= g[i][j];
            }
        }
        int thisw= g.length;
        int thish= g[0].length;
        int w= kenel.length;
        int h= kenel[0].length;
        int [][]thisG = null;
        if((kenel.length%2== 0)&&(kenel[0].length%2== 0))
            thisG= new Reflection().PadImage(g1, kenel.length+1, kenel[0].length+ 1);
        if((kenel.length%2== 0)&&(kenel[0].length%2== 1))
            thisG= new Reflection().PadImage(g1, kenel.length+1, kenel[0].length);
        if((kenel.length%2== 1)&&(kenel[0].length%2== 0))
            thisG= new Reflection().PadImage(g1, kenel.length, kenel[0].length+ 1);
        if((kenel.length%2== 1)&&(kenel[0].length%2== 1))
            thisG= new Reflection().PadImage(g1, kenel.length, kenel[0].length);
        for(int i= 0; i< thisG.length; i++) {
            for(int j= 0; j< thisG[0].length;
                j++){ if (thisG[i][j]!= pix){
                    thisG[i][j]= 0; // filter
                }
            }
        }
        initTemp(thisG);
        for(int
            i=w/2;i<thisw+w/2;i++){ for(int
            j=h/2;j<thish+h/2;j++){
                int last=0;
                for(int
```

```
                    p=0;p<w;p++){ for(int
                    q=0;q<h;q++){
                        int temp = 0;
                        if(kenel[p][q]==1){
                            if(thisG[i-w/2+p][j-
                                h/2+q]>0){ temp=1;
                            }
                        }
                        if(kenel[p][q]==0){
                            if(thisG[i-w/2+p][j-
                                h/2+q]==0){ temp=1;
                            }
                        }
                        if(kenel[p][q]==-
                            1){ temp=1;
                        }
                        last=last+temp;
                    }
                }
                if
                    (last==w*h){ tempG[i][j]
                    =thisG[i][j];
                }
            }
        }
        return tempG;
    }
}
```

==========

#函数名：霍夫变换

#函数思想作者：Bubble Chamber 1959

#函数功能作者：罗瑶光

#函数用途：用于雷达，气象，图片的角度和曲度值标注。

#函数解析：通过类似傅里叶的公式套用将像素与坐标2个维度进行整合，得到一个观测矩阵。

```
package imageProcessor;
public class HoughTransform
    { public int [][]HTMatrix;
    public void HoughTransformLoop(int[][] g, int pix, int scale)
        { for(int i=0; i< g.length; i++){
            for(int j=0; j< g[0].length;
                j++){ if(g[i][j]== pix){
                    for(int p=0; p<360; p++){
                        double sita= p* Math.PI/ 360;
                        int psita=(int) (i* Math.cos(sita)+j*  Math.sin(sita));
                        if (psita<0) psita=0;
                        HTMatrix[psita][p]=HTMatrix[psita][p]+scale;
                    }
```

```
                }
            }
        }
    }
    public void initHTMatrix(int[][] g)
        { int max= 0;
        for(int i= 0; i<g.length; i++){
            for(int j= 0; j< g[0].length;
                j++){ for(int p= 0; p< 360;
                p++){
                    double sita= p* Math.PI/ 360;
                    int temp= (int) (i* Math.cos(sita)+  j* Math.sin(sita));
                    if (temp>= max) {
                        max= temp;
                    }
                }
            }
        }
        HTMatrix= new int[max+1][360];
        for(int p=0; p<360; p++) {
            for(int psita= 0; psita< max+ 1; psita++)
                { HTMatrix[psita][p]= 0;
            }
        }
    }
}
```

==========

#函数名：拉普拉斯过滤

#函数思想作者：拉普拉斯

#函数功能作者：罗瑶光

#函数用途：用于超快速的得到图片中像素边缘。

#函数解析：通过拉普拉斯算子进行矩阵卷积处理。

```
package imageProcessor;
import java.io.IOException;
public class Laplacian {
    public int[][] Processor( int[][] g) throws IOException
        { int[][] refG =new Reflection().PadImage(g, 3, 3);
        int[]size = {g.length,g[0].length};
        double[][] Gx = new double [size[0]][size[1]];
        int[][] gxk = new int[][]{
            { 0, 1, 0},
            { 1,-4, 1},
            { 0, 1, 0}};
            //GROUP OPERATION
            Gx= new GroupOperator().GO(gxk, refG, size);
            for(int i=0;i<Gx.length;i++){
                for(int j=0;j<Gx[0].length;j++){
```

```
                    g[i][j]=(int) (Gx[i][j]/8+127);
                }
            }
            return  new CheckRange().Processor(g);
    }
}
```

==========

#函数名：掩码归零
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：用于基于掩码的矩阵快速归零
#函数解析：矩阵遍历条件比较归零。

```
package imageProcessor;
import java.io.IOException;
public class Mask {
    public int[][] Processor(int[][] mag,int[][]dir) throws IOException
        { for(int i =0;i< mag.length;i++){
            for(int j =0;j<
                mag[0].length;j++){ if
                (mag[i][j]==0){
                    dir[i][j]=0;
                }
            }
        }
        return dir;
    }
}
```

==========

#函数名：均值过滤
#函数思想作者：无
#函数功能作者：罗瑶光
#函数用途：用于像素平滑过滤
#函数解析：通过每一个像素的周围算子进行求和的平均值来做平滑卷积过滤。

```
package imageProcessor;
import java.awt.image.BufferedImage;
import java.io.IOException;
public class Median {
    public int[][] Processor(int[][] g,int d,int e) throws IOException
        { int[][] refG =new Reflection().PadImage(g, d, e);
        int[] neib=new int[d*e];//for sort
        for(int i=d/2;i<g.length+d/2;i++){
            for(int j=e/2;j<g[0].length+e/2;j++)
                { int q=0;
                for(int k=0;k<d;k++)
                    { for(int l=0;l<e;l++)
                    {
```

```
                        //find 3*3
                        neib[q++]=refG[i+k-d/2][j+l-e/2];
                        //sort 3*3 bbsort
                    }}
                for(int o=0;o<9;o++)
                    { for(int
                    p=0;p<9;p++){
                        if (neib[o] > neib[p])
                            { int temp = neib[o];
                            neib[o] = neib[p];
                            neib[p] = temp;
                        }
                    }
                }
                //get median
                g[i-d/2][j-e/2]=neib[(d*e)/2];// 4 is middle valueof sort 9
            }
        }
        return new CheckRange().Processor(g);
    }
    public BufferedImage Processor(BufferedImage lygimage, int d, int e) throws IOException
        { int r[][]=new ReadWritePng().REDpngRead(lygimage);
        int g[][]=new ReadWritePng().GRNpngRead(lygimage);
        int b[][]=new ReadWritePng().BLUpngRead(lygimage);
        //r[][]
        r=Processor(r,d,e);
        //g[][]
        g=Processor(g,d,e);
        //b[][]
        b=Processor(b,d,e);
        //r[][]g[][]b[][] to image
        lygimage=new ReadWritePng().createBufferImage(r,g,b);
        return lygimage;
    }
}
```

#函数名：像素细化膨胀处理中的逼近非连接处理

#函数思想作者：无

#函数功能作者：罗瑶光

#函数用途：像素在细化过程中的膨胀逼近非连接处理。

```
package imageProcessor;
public class Opening {
    public int[][]
        Processor(int[][]g,int[][]kenel){ int[][]
        opening=new Erosion().Processor(g,kenel);
        opening=new Dilation().Processor(opening, kenel);
        return opening;
    }
```

```
}
==============================================
```

#函数名：像素矩阵的文件读写

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：像素矩阵的文件读写

```
package imageProcessor;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class ReadWritePng {
    public int h;
    public int w;
    public void writePNG(String args, int[][] outmag) throws
        IOException{ BufferedImage image = new BufferedImage(outmag[0].length,
        outmag.length
                , BufferedImage.TYPE_INT_RGB);
        for (int i = 0; i < image.getHeight(); ++i)
            { for (int j = 0; j < image.getWidth(); ++j)
            {
                int val = (int) outmag[i][j];
                int pixel = (val << 16) | (val << 8) | (val);
                image.setRGB(j, i, pixel);
            }
        }
        String pathBin = args;//output path
        File outputBin = new File(pathBin);
        ImageIO.write(image, "png", outputBin);
    }
    public ReadWritePng(){
    }
    public int[][] GRNpngRead(String args) throws IOException
        { BufferedImage image = ImageIO.read(new File(args));
        // extract R, G, B values
        h = image.getHeight();
        w = image.getWidth();
        System.out.println(h+"&&"+w);
        int[][] g = new int[h][w];
        for (int i = 0; i < h; i++){
            for (int j = 0; j < w; j++){
                g[i][j] = image.getRGB(j, i) >> 8 & 0xFF;
            }
        }
        return g;
    }
    public int[][] GRNpngRead(BufferedImage image) throws IOException {
```

```
    // extract R, G, B values
    h = image.getHeight();
    w = image.getWidth();
    System.out.println(h+"&&"+w);
    int[][] g = new int[h][w];
    for (int i = 0; i < h; i++){
        for (int j = 0; j < w; j++){
            g[i][j] = image.getRGB(j, i) >> 8 & 0xFF;
        }
    }
    return g;
}
public int[][] REDpngRead(String args) throws IOException
    { BufferedImage image = ImageIO.read(new File(args));
    // extract R, G, B values
    h = image.getHeight();
    w = image.getWidth(); int[][]
    r = new int[h][w]; for (int
    i = 0; i < h; i++){
        for (int j = 0; j < w; j++){
            r[i][j] = image.getRGB(j, i) >> 16 & 0xFF;
        //x,y coordinates opposite to array
        }
    }
    return r;
}
public int[][] REDpngRead(BufferedImage image) throws IOException {
    // extract R, G, B values
    h = image.getHeight();
    w = image.getWidth(); int[][]
    r = new int[h][w]; for (int
    i = 0; i < h; i++){
        for (int j = 0; j < w; j++){
            r[i][j] = image.getRGB(j, i) >> 16 & 0xFF;
        //x,y coordinates opposite to array
        }
    }
    return r;
}
public int[][] BLUpngRead(String args) throws IOException
    { BufferedImage image = ImageIO.read(new File(args));
    // extract R, G, B values
    h = image.getHeight();
    w = image.getWidth();
    int[][] b = new int[h][w];
    for (int i = 0; i < h; i++){
        for (int j = 0; j < w; j++){
```

```
                b[i][j] = image.getRGB(j, i) & 0xFF;
            }
        }
        return b;
    }
    public int[][] BLUpngRead(BufferedImage image) throws IOException {
        // extract R, G, B values
        h = image.getHeight();
        w = image.getWidth(); int[][]
        b = new int[h][w]; for (int
        i = 0; i < h; i++){
            for (int j = 0; j < w; j++){
                b[i][j] = image.getRGB(j, i) & 0xFF;
            }
        }
        return b;
    }
    public int[] sizeHW(String args) throws IOException
        { BufferedImage image = ImageIO.read(new
        File(args)); int size[] =new int[2];
        size[0]= image.getHeight();
        size[1]= image.getWidth();
        return size;
    }
    public BufferedImage createBufferImage(int[][] r, int[][] g, int[][] b)
        { BufferedImage image = new BufferedImage(r[0].length, r.length
                , BufferedImage.TYPE_INT_RGB);
        for (int i = 0; i < image.getHeight(); ++i) {
            for (int j1 = 0; j1 < image.getWidth(); ++j1)
                { int rr = (int) r[i][j1];
                int gg = (int) g[i][j1];
                int bb = (int) b[i][j1];
                int pixel = (rr << 16) | (gg << 8) | (bb);
                image.setRGB(j1, i, pixel);
            }
        }
        return image;
    }
}
```

==========

#函数名：像素矩阵的文件读写

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：像素矩阵的文件读写

```
package imageProcessor;
import imageProcessor.ReadWritePng;
import java.io.IOException;
```

```
public class RegionGet {
    int[] pix;
    public int []new_region;
    public int scale;
    public RegionGet(int[][] g) throws
        IOException{ int n=0;
        pix=new int[256];
        //统计像素数
        for (int
            i=0;i<g.length;i++){ for(int
            j=0;j<g[0].length;j++) {
                pix[g[i][j]]++;
            }
        }
        //统计像素段总数
        for(int i=0;i<256;i++){
            if(pix[i]>0){
                 n++;
            }
        }
        System.out.println(n);
        new_region=new int[n];
        n=0;
        //有效像素段序列化
        for(int i = 0; i < 256;
            i++){ if(pix[i]>0) {
                new_region[n++] = i;
            }
        }
        System.out.println("value"+n);
        scale=new_region.length-1;
        //return new_region;
    }
    //将像素段有效的等比放大显色成人类肉眼能分辨的图片
    public void buildGraph(int[][] g,String output) throws
        IOException{ int[][]temp=new int[g.length][g[0].length];
        for(int q=0;q<g.length;q++)
            { for(int p=0;p<g[0].length;p++)
            {
                temp[q][p]=g[q][p];
            }
        }
        for(int q=0;q<g.length;q++)
            { for(int
            p=0;p<g[0].length;p++){
                for(int i=0;i<scale;i++){
                    if
```

```
                        (temp[p][q]==new_region[i])
                        { temp[p][q]=255/scale*i;
                    }}
        new ReadWritePng().writePNG(output, temp);
    }
}
```

==========

#函数名：索贝尔过滤算法
#函数思想作者：R.Fletche
#函数功能作者：罗瑶光
#函数用途：用于图形的数据边缘识别
#函数解析：通过索贝尔算子做卷积(dir)边缘过滤和（mag）梯度过滤，然后两种对象2选1输出。

```
package imageProcessor;
import java.io.IOException;
public class Sobel {
    public int[][] Processor( int[][] g,int choice) throws IOException
        { int[][] refG = new Reflection().PadImage(g, 3, 3);
        int[]size = {g.length,g[0].length};
        double[][] Gx= new double [size[0]][size[1]];
        double[][] Gy= new double [size[0]][size[1]];
        int[][] gxk= new int[][]{{-1, 0, 1},{-2, 0, 2},{-1, 0, 1}};;
        int[][] gyk= new int[][]{{-1,-2,-1},{ 0, 0, 0},{ 1, 2, 1}};
        //GROUP OPERATION
        Gx= new GroupOperator().GO(gxk, refG, size);
        Gy= new GroupOperator().GO(gyk, refG, size);
        //MAG operation
        int[][] outmag= mag(Gx,Gy,size);
        //DIR operation
        int[][] outdir= dir(Gx,Gy,size);
        switch (choice){
            case 1: return outmag;
            case 2: return outdir;
            default: return null;
        }
    }
    private int[][] mag(double[][] gx, double[][] gy, int[]size) throws
        IOException{ double[][] mag= new double[size[0]][size[1]];
        int[][] outmag= new int[size[0]][size[1]];
        for(int i= 0; i<size[0]; i++){
            for(int j= 0; j<size[1]; j++){
                mag[i][j]= Math.sqrt(Math.pow(gx[i][j], 2)+ Math.pow(gy[i][j], 2));
                outmag[i][j]= (int)(mag[i][j]/(1020* Math.sqrt(2))* 255);
            }
        }
        return  new CheckRange().Processor(outmag);
    }
    private int[][] dir(double[][] gx,double[][] gy,int[]size) throws
```

```
        IOException{ double[][] dir = new double[size[0]][size[1]];
        int[][] outdir = new int[size[0]][size[1]];
        for(int i = 0;i<size[0];i++){
            for(int j = 0;j<size[1];j++){
                dir[i][j] = Math.atan2(gy[i][j], gx[i][j]);
                outdir[i][j] = (int)((dir[i][j]+Math.PI)/(2*Math.PI)*32)*8;
                if(outdir[i][j]>255){
                    outdir[i][j]=255;
                }
            }
        }
        return  new CheckRange().Processor(outdir);
    }
}
```

==========

#函数名：像色拉伸算法

#函数思想作者：无法找到

#函数功能作者：罗瑶光

#函数用途：用于图形的色彩拉伸

#函数解析：将0~255的像素中某个段进行重新按0~255比例拉伸。

```
package imageProcessor;
import java.awt.image.BufferedImage;
import java.io.IOException;
public class Strech {
    public int[][] Processor(int[][] g,double d,double e)
            throws IOException {
        int histgram[] = new int[256];
        for (int i= 0; i< g.length; i++) {
            for (int j= 0; j< g[0].length; j++) {
                ++histgram[g[i][j]];
            }
        }
        float p = 0;
        int br = 0, dr = 0;
        for(int i=0; i<256; i++) {
            p=p+histgram[i];
            if(p > d*
                g.length*g[0].length){ dr= i;
                break;
            }
        }
        System.out.println("dr:"+dr);
        p = 0;
        for(int i=0; i<256;
            i++){ p= p+
            histgram[i];
            if(p > e*
```

```
                g.length*g[0].length){ br=i;
                break;
            }
        }
        System.out.println("br:"+br);
        int out[][]= new int[g.length][g[0].length];
        for (int i= 0; i< g.length; i++) {
            for (int j= 0; j< g[0].length; j++)
                { out[i][j]= (g[i][j]-dr)*255/(br-
                dr);
            }
        }
        return new CheckRange().Processor(out);
    }
    public BufferedImage Processor(BufferedImage lygimage, double d
            , double e) throws IOException {
        int r[][]= new ReadWritePng().REDpngRead(lygimage);
        int g[][]= new ReadWritePng().GRNpngRead(lygimage);
        int b[][]= new ReadWritePng().BLUpngRead(lygimage);
        //r[][]
        r=Processor(r,d,e);
        //g[][]
        g=Processor(g,d,e);
        //b[][]
        b=Processor(b,d,e);
        //r[][]g[][]b[][] to image
        lygimage=new ReadWritePng().createBufferImage(r,g,b);
        return lygimage;
    }
}
```

==========

#函数名：2分极化算法
#函数思想作者：无法找到
#函数功能作者：罗瑶光
#函数用途：用于图形的色彩极化区分
#函数解析：将0~255的像素中按条件分离成 0 和255.

```
package imageProcessor;
public class Threshold {
    public int[][]  Processor (int[][] g,int Td){
        // -- prepare output image for subtract the mean
        for (int i= 0; i< g.length; ++i) {
            for (int j= 0; j< g[0].length; ++j)
                { if( g[i][j]>Td){
                    g[i][j]=255;
                }
                else{
                    g[i][j] = 0; // sho
```

```
                }
            }
        }
        return g;
    }
}
```

==========

#函数名：卷积均值算法

#函数思想作者：Yann LeCun

#函数功能作者：罗瑶光

#函数用途：用于处理矩阵的列平均值比率计算

#函数解析：通过矩阵的列的算子和来求其所在的列的概率比率。

```
package matrixProcessor;
public class CnnMeasure{
    public double[][] getCnnMeansure(double inputs[][]){
        double[][] output= new double[inputs.length][inputs[0].length];
        double[] Kernel= new double[inputs[0].length];
        for(int j = 0; j<inputs[0].length;j++)
            { double sum=0;
            for(int k = 0; k<inputs.length;k++)
                { sum+= inputs[k][j];
            }
            Kernel[j]= sum;
        }
        for(int i = 0; i<inputs.length;i++)
            { for(int j = 0; j<inputs[0].length;j++)
            {
                output[i][j] = inputs[i][j]/Kernel[j];
            }
        }
        return output;
    }
}
```

==========

#函数名：卷积均值算法针对我的deta parser项目

#函数思想作者：Yann LeCun

#函数功能作者：罗瑶光

#函数用途：用于处理矩阵的列平均值比率计算

#函数解析：通过矩阵的列的算子和来求其所在的列的概率比率。

#备注：从7开始是因为我的detaParser对应矩阵数据前6个存放了非计算数据。

```
package NLPProcessor;
import java.io.IOException;
public class DETA_ANN_HMM {
    public String[][] summingProcessor(String[][] inputNLP) throws IOException
    , InstantiationException, IllegalAccessException
        { String[][] outNLP = inputNLP.clone();
        for(int i = 7; i < inputNLP[0].length; i++)
```

```
            { for(int j = 0; j < inputNLP.length; j++)
            {
                double sum = 0;
                for(int k = 0;k < inputNLP.length; k++)
                    { sum +=
                    Double.valueOf(inputNLP[k][i]);
                }
                //System.out.println(sum);
                outNLP[j][i] = "" + Double.valueOf(inputNLP[j][i])/sum;
            }
        }
        return outNLP;
    }
}
```

==========

#函数名：卷积深度均值组合算法针对我的deta parser读心术重心项目
#函数思想作者：<lwa: Geoffrey Hinton> <cnn: Yann LeCun> <整体函数：罗瑶光>
#函数功能作者：罗瑶光
#函数用途：用于deta读心术的词汇重心比率计算
#函数解析：将ann的数据进行rnn的卷积做LWA卷积处理。
#备注：从7开始是因为我的detaParser对应矩阵数据前6个存放了非计算数据。

```
package NLPProcessor;
import java.io.IOException;
public class DETA_DNN {
    public String[][] summingProcessor(String[][] ann, String[][] rnn) throws IOException
    , InstantiationException, IllegalAccessException {
        //SUM OF LWA DNN
        //1 DNN AOP OF DNN
        //IDE= "亲密，离散，位置"
        String[][] dnn = new String[rnn.length][4];
        for(int i = 0; i < rnn.length; i++) {
            dnn[i][0] = rnn[i][0];
            double important = 0;
            for(int j = 0; j < rnn.length; j++) {
                important += Double.parseDouble(rnn[j][2]);
            }
            dnn[i][1] = "" + Math.sqrt(important * Double.parseDouble(rnn[i][1]));
        }
        //2 DNN CORRELATION LWA
        for(int i = 0; i < rnn.length; i++)
            { double correlation = 0;
            double sumOfPow = 0;
            for(int j = 0; j < rnn.length; j++) {
                sumOfPow +=  Double.parseDouble(rnn[j][3]);
                //sumOfPow += Math.abs(Double.parseDouble(rnn[i][3]) - Double.parseDouble(rnn[j][3]));
            }
        //2 sumOfPow = Math.abs(Double.parseDouble(rnn[i][3]) * (double)rnn.length - sumOfPow);
```

```
        //  sumOfPow /= rnn.length;
            sumOfPow = Math.abs(Double.parseDouble(rnn[i][3]) - sumOfPow/rnn.length);//3
            correlation =  Double.parseDouble(dnn[i][1]) * sumOfPow;
            dnn[i][2] = "" + Math.sqrt(correlation);
        }
        //3 DNN and summing ANN combination
        for(int i = 0; i < ann.length; i++) {
            double summingANN = 0;
            double combination = 0;
            for(int j = 7; j < ann[0].length; j++)
                { summingANN +=
                Double.parseDouble(ann[i][j]);
            }
            combination = summingANN *  Double.parseDouble(dnn[i][2]);
            dnn[i][3] = "" + combination;
        }
        return dnn;
    }
}
```

==========

#函数名：2分发搜索
#函数思想作者：无法找到
#函数功能作者：罗瑶光
#函数用途：用于快速2分查找
#函数解析：通过2的指数方拆分进行非线性快速查找

```
package searchProcessor;
public class BinarySearch{
    public boolean search(int [] array,int n)
        { int low = 0;
        int high=array.length-1;
        while(low<high){
            int mid=(low+high)/2;
            if(array[mid]==n) {
                return true;
            }else {
                if(n<array[mid])
                    { high=mid-1;
                }else{
                    low=mid+1;
                }
            }
        }
        return false;
    }
}
```

==========

#函数名：广度8叉数搜索验证算法

#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：用于快速8叉查找
#函数解析：通过递归来进行8叉树广度查找。

```
package searchProcessor;
import sortProcessor.Leaf;
//import com.sun.image.codec.jpeg.JPEGCodec;
//import com.sun.image.codec.jpeg.JPEGImageDecoder;
public class BreadthRun extends Thread{
    public boolean result;
    Leaf temp1;
    int n;
    public boolean end=false;
    public Node head,back;
    public BreadthRun(Leaf root, int n1)
        { temp1=root;
        n=n1;
    }
    @Override
    public void run(){
        result=false;
        check(temp1);
        while(head!=null){
            pop();
        }
        end=true;
    }
    @SuppressWarnings("unused")
    private void pop() {
        if(head!=null){ if(head.val
            ue.sort==0){
                if(head.value.has01==1){
                    if(n ==
                        head.value.value[0]){ res
                        ult=true;
                        end=true;
                    }
                }
                if(head.value.has02==1){
                    if(n ==
                        head.value.value[1]){ res
                        ult=true;
                        end=true;
                    }
                }
                if(head.value.has03==1){
                    if(n ==
```

```
            head.value.value[2]){ res
            ult=true;
            end=true;
        }
    }
    if(head.value.has04==1){
        if(n ==
            head.value.value[3]){ res
            ult=true;
            end=true;
        }
    }
    if(head.value.has05==1){
        if(n ==
            head.value.value[4]){ res
            ult=true;
            end=true;
        }
    }
    if(head.value.has06==1){
        if(n ==
            head.value.value[5]){ res
            ult=true;
            end=true;
        }
    }
    if(head.value.has07==1){
        if(n == head.value.value[6]){
            result=true;
            end=true;
        }
    }
    head.value.sort=1;
    if(head.value.01!=null&&head.value.01.sort==0)
        push(head.value.01);
    if(head.value.02!=null&&head.value.02.sort==0)
        push(head.value.02);
    if(head.value.03!=null&&head.value.03.sort==0)
        push(head.value.03);
    if(head.value.04!=null&&head.value.04.sort==0)
        push(head.value.04);
    if(head.value.05!=null&&head.value.05.sort==0)
        push(head.value.05);
    if(head.value.06!=null&&head.value.06.sort==0)
        push(head.value.06);
    if(head.value.07!=null&&head.value.07.sort==0)
        push(head.value.07);
```

```
                if(head.value.08!=null&&head.value.08.sort==0)
                    push(head.value.08);
            }
        }
        Node vv=head;
        head=head.next;
        vv=null;
    }
    private void check(Leaf temp)
        { if(temp!=null){
            if(temp.sort==0){ if(t
                emp.has01==1){
                    if(n ==
                        temp.value[0]){ res
                        ult=true; end=true;
                    }

                }
                if(temp.has02==1){
                    if(n ==
                        temp.value[1]){ res
                        ult=true; end=true;
                    }
                }
                if(temp.has03==1){
                    if(n ==
                        temp.value[2]){ res
                        ult=true; end=true;
                    }
                }
                if(temp.has04==1){
                    if(n ==
                        temp.value[3]){ res
                        ult=true; end=true;
                    }
                }
                if(temp.has05==1){
                    if(n ==
                        temp.value[4]){ res
                        ult=true; end=true;
                    }
                }
                if(temp.has06==1){
                    if(n ==
                        temp.value[5]){ res
                        ult=true; end=true;
                    }
```

```
                }
                if(temp.has07==1){
                    if(n ==
                        temp.value[6]){ res
                        ult=true; end=true;
                    }
                }
                temp.sort=1;
                if(temp.01!=null&&temp.01.sort==0)
                    push(temp.01);
                if(temp.02!=null&&temp.02.sort==0)
                    push(temp.02);
                if(temp.03!=null&&temp.03.sort==0)
                    push(temp.03);
                if(temp.04!=null&&temp.04.sort==0)
                    push(temp.04);
                if(temp.05!=null&&temp.05.sort==0)
                    push(temp.05);
                if(temp.06!=null&&temp.06.sort==0)
                    push(temp.06);
                if(temp.07!=null&&temp.07.sort==0)
                    push(temp.07);
                if(temp.08!=null&&temp.08.sort==0)
                    push(temp.08);
            }
        }
    }
    private void push(Leaf cur)
        { if(head==null){
            head=new Node();
            head.value=cur;
            back=head;
        }else{
            Node curr=new Node();
            curr.value=cur;
            back.next=curr;
            back=curr;
        }
    }
}
```

==========

#函数名：广度搜索验证算法

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型

```
package searchProcessor;
import sortProcessor.Leaf;
```

```
public class BreadthTreeSearch{
    public boolean search(Leaf root,int n1)
        { BreadthRun pr= new
        BreadthRun(root,n1); Thread t= new
        Thread(pr);
        t.run();
        while(true)
                    {
         if(pr.end){
            break;
            }
        }
        boolean ans=pr.result;
        t=null;
        pr=null;
        return ans;
    }
}
```

==========

#函数名：深度搜索验证算法
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package searchProcessor;
import sortProcessor.Leaf;
//import com.sun.image.codec.jpeg.JPEGCodec;
//import com.sun.image.codec.jpeg.JPEGImageDecoder;
public class DepthRun extends Thread{
    public boolean result;
    Leaf temp1;
    int n;
    public boolean end=false;
    public DepthRun(Leaf root, int n1)
        { temp1=root;
        n=n1;
    }
    @Override
    public void
        run(){ result=
        false;
        check(temp1);
        end=true;
    }
    private void check(Leaf temp)
        { if(temp!=null){
            if(temp.sort==0){ if(t
                emp.has01==1){
```

```
    if(n ==
        temp.value[0]){ res
        ult=true; end=true;
    }

}
if(temp.has02==1){
    if(n ==
        temp.value[1]){ res
        ult=true; end=true;
    }
}
if(temp.has03==1){
    if(n ==
        temp.value[2]){ res
        ult=true; end=true;
    }
}
if(temp.has04==1){
    if(n ==
        temp.value[3]){ res
        ult=true; end=true;
    }
}
if(temp.has05==1){
    if(n ==
        temp.value[4]){ res
        ult=true; end=true;
    }
}
if(temp.has06==1){
    if(n ==
        temp.value[5]){ res
        ult=true; end=true;
    }
}
if(temp.has07==1){
    if(n ==
        temp.value[6]){ res
        ult=true;
        end=true;
    }
}
temp.sort=1;
if(temp.01!=null&&temp.01.sort==0)
    check(temp.01);
if(temp.02!=null&&temp.02.sort==0)
```

```
                    check(temp.02);
                if(temp.03!=null&&temp.03.sort==0)
                    check(temp.03);
                if(temp.04!=null&&temp.04.sort==0)
                    check(temp.04);
                if(temp.05!=null&&temp.05.sort==0)
                    check(temp.05);
                if(temp.06!=null&&temp.06.sort==0)
                    check(temp.06);
                if(temp.07!=null&&temp.07.sort==0)
                    check(temp.07);
                if(temp.08!=null&&temp.08.sort==0)
                    check(temp.08);
            }
        }
    }
}
```

```
==============================================
```

```
package searchProcessor;
import sortProcessor.Leaf;
public class DepthTreeSearch{
    public boolean search(Leaf root,int n1)
        { DepthRun pr=new DepthRun(root,n1);
        Thread t=new Thread(pr);
        t.run();
        while(true)  {
             if(pr.end){
                 break;
            }
        }
        boolean ans=pr.result;
        t=null;
        pr=null;
        return ans;
    }
}
```

```
==============================================
```

#函数名：线性搜索验证算法
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package searchProcessor;
public class LinerSearch{
    public boolean search(int [] array,int n)
        { for(int i=0;i<array.length;i++){
            if(array[i]==n)
                return true;
```

```
        }
        return false;
    }
}
```

==========

```
package searchProcessor;
import sortProcessor.Leaf;
public class Node {
  public Leaf value;
  public Node next;
}
```

==========

```
package searchProcessor;
import sortProcessor.Leaf;
//import com.sun.image.codec.jpeg.JPEGCodec;
//import com.sun.image.codec.jpeg.JPEGImageDecoder;
public class PreorderRun extends Thread{
    public boolean result;
    Leaf temp1;
    int n;
    public boolean end=false;
    public PreorderRun(Leaf root, int n1)
    { temp1=root;
    n=n1;
    }
    @Override
    public void
        run(){ result=
        false;
        check(temp1);
        end=true;
    }
    private void check(Leaf temp)
        { if(temp!=null){
            check(temp.01);
            if(temp.has01==1){
                if(n ==
                    temp.value[0]){ res
                    ult=true; end=true;
                }
            }
            check(temp.02);
            if(temp.has02==1){
                if(n ==
                    temp.value[1]){ res
                    ult=true;
                    end=true;
```

```
                }
            }
            check(temp.03);
            if(temp.has03==1){
                if(n ==
                    temp.value[2]){ res
                    ult=true; end=true;
                }
            }
            check(temp.04);
            if(temp.has04==1){
                if(n ==
                    temp.value[3]){ res
                    ult=true; end=true;
                }
            }
            check(temp.05);
            if(temp.has05==1){
                if(n ==
                    temp.value[4]){ res
                    ult=true; end=true;
                }
            }
            check(temp.06);
            if(temp.has06==1){
                if(n ==
                    temp.value[5]){ res
                    ult=true; end=true;
                }
            }
            check(temp.07);
            if(temp.has07==1){
                if(n ==
                    temp.value[6]){ res
                    ult=true; end=true;
                }
            }
            check(temp.08);
        }
    }
}
```

==========

#函数名：前序树搜索验证算法

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型

```
package searchProcessor;
```

```
import sortProcessor.Leaf;
public class PreorderTreeSearch{
    public boolean search(Leaf root,int n1)
        { PreorderRun pr=new
        PreorderRun(root,n1); Thread t=new
        Thread(pr);
        t.run();
        while(true){
             if(pr.end){
                 break;
            }
        }
        boolean ans=pr.result;
        t=null;
        pr=null;
        return ans;
    }
}
```

==========

#函数名：随机搜索验证算法
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package searchProcessor;
public class RandomSearch{
    public boolean search(int [] array,int n)
        { int ran[] = new int[array.length];
        java.util.Random r = new java.util.Random();
        int i=0;
        while(i<array.length){
            int j = r.nextInt(array.length);
            if(ran[j] == 0) {
                if(array[j] ==
                    n){ return
                    true;
                }else{
                    ran[j]=1;
                    i+=1;
                }
            }
        }
        return false;
    }
}
```

==========

#函数名：2分法排序算法
#函数思想作者：罗瑶光

#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
public class BinarySort{
    public int[] sort(int [] a) {
        int ps = 0;
        int pb = 0;
        int fs=0;
        int fb=0;
        int e=a.length;
        for(int i=0;i<e;i++,e--
            ){ int s=a[i];
            int b=a[i];
            for(int
                j=i;j<e;j++){ if(
                s>a[j]){
                    s=a[j];
                    ps=j;
                    fs=1;
                }
                if(b<a[j]){
                    b=a[j];
                    pb=j;
                    fb=1;
                }
            }
             if(fs==1){
                fs=0;
                int temp=a[i];
                a[i]=s;
                a[ps]=temp;
            }
            if(fb==1){
                int temp=a[e-1];
                a[e-1]=b;
                a[pb]=temp;
            }
        }
        return a;
    }
}
```

==========

#函数名：2叉数排序算法
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
```

```
public class BTreeSort{
    public Leaf root;
    public Leaf heap;
    int c;
    int a1[];
    public int[] sort(int [] a) {
        //make tree
        c=0;
        a1=new int[a.length];
        if(root==null){
            root=new Leaf();
            root.value[0]=a[0];
            root.has01=1;
        }
        for(int
            i=1;i<a.length;i++){ hea
            p=root; addleaf(a[i]);
        }
        heap=root;
        check(heap);
        return a1;
    }
    private void check(Leaf temp)
        { if(temp!=null){
            check(temp.01);
            a1[c]=temp.value[0];
            c+=1;
            check(temp.02);
        }
    }
    private void addleaf(int i)
        { if(i<=heap.value[0]){
            if(heap.01==null){ heap.
                01=new Leaf();
                heap=heap.01;
                heap.value[0]=i;
                root.has01=1;
                return;
            }
            else{
                heap=heap.01;
                addleaf(i);
            }
        }
        else{
            if(heap.02==null){ heap.
                02=new Leaf();
```

```
                heap=heap.02;
                heap.value[0]=i;
                root.has01=1;
                return;
            }else{
                heap=heap.02;
                addleaf(i);
            }
        }
    }
    public Leaf root(int[] a)
        { if(root==null){
            root=new Leaf();
            root.value[0]=a[0];
            root.has01=1;
        }for(int
            i=1;i<a.length;i++){ hea
            p=root; addleaf(a[i]);
        }
        return root;
    }
}
```

==========

#函数名：堆排序算法
#函数思想作者：无
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
public class Heap_1D_Sort{
    int a[],s;
    public int[] sort(int [] array)
        { a=new int[array.length+1];
        s=array.length;
        for(int i=1;i<a.length;i++)
            a[i]=array[i-1];
        find(1);
        while(s>1){
            int t=a[1];a[1]=a[s];a[s]=t;
            s-=1;
            find(1);
        }
        for(int i=1;i<a.length;i++)
            array[i-1]=a[i];
        return array;
    }
    private  void  find(int  i)
        {                  int
```

```
        il=2*i,ir=2*i+1,t;
        if(il<=s&&ir<=s)
        {find(il);find(ir);}
        else if(il<=s&&ir>s)
        {find(il);}
        else if(ir<=s&&il>s)
        {find(ir);}
        if(il<=s)
            if((a[il]>a[i]))
            {t=a[i];a[i]=a[il];a[il]=t;find(il);}
        if(ir<=s)
            if((a[ir]>a[i]))
            {t=a[i];a[i]=a[ir];a[ir]=t;find(ir);}
    }
}
```

==========

#函数名：插入排序算法
#函数思想作者：无
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
public class InsertionSort{
    public int[] sort(int [] array)
        { int j;
        for(int
            i=1;i<array.length;i++){ j=i;
            while(j>=1){
                if(array[j]<array[j-
                    1]){ int
                    temp=array[j];
                    array[j]=array[j-1];
                    array[j-1]=temp;
                }
                j-=1;
            }
        }
        return array;
    }
    public int[] sort(int [] array, int n)
        { int j;
        for(int
            i=1;i<n;i++){ j=
            i; while(j>=1){
                if(array[j]<array[j-
                    1]){ int
                    temp=array[j];
                    array[j]=array[j-1];
```

```
                    array[j-1]=temp;
                }
                j-=1;
            }
        }
        return array;
    }
}
```

==========

```
package sortProcessor;
public class Leaf{
    public int sort=0;
    public Leaf 01;
    public Leaf 02;
    public Leaf 03;
    public Leaf 04;
    public Leaf 05;
    public Leaf 06;
    public Leaf 07;
    public Leaf 08;
    public int value[]=new int[7];
    public int has01=0;
    public int has02=0;
    public int has03=0;
    public int has04=0;
    public int has05=0;
    public int has06=0;
    public int has07=0;
}
```

==========

#函数名：链表排序算法
#函数思想作者：无
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
class node {
    public node next;
    public int value;
}
public class
    LinkSort{ public
    node heap; public
    node first;
    public int[] sort(int [] a)
        { for(int
        i=0;i<a.length;i++){
            lyg(a[i]);
```

```
        }
        for(int
            i=0;i<a.length;i++){ a[i]
            =first.value;
            if(first.next!=null)
                first=first.next;
        }
        return a;
    }
    private void lyg(int i)
        { if(heap==null){
            heap=new node();
            heap.value=i;
            first=heap;
            return;
        }
        heap=first;
        if(i<=heap.value){
            node temp=new node();
            temp.value=i;
            temp.next=heap;
            first=temp;
            return;
        }
        while(heap.next!=null){ if(i>=heap.value&&i<=hea
            p.next.value){
                node temp=new node();
                temp.value=i;
                temp.next=heap.next;
                heap.next=temp;
                return;
            }
            heap=heap.next;
        }
        node temp=new node();
        heap.next=temp;
        temp.value=i; return;
    }
}
```

==========

#函数名：仿照2叉树做数组排序算法

#函数思想作者：2叉数原理

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型

```
package sortProcessor;
import java.util.ArrayList;
import java.util.List;
```

```
//import parserProcessor.timeCheck;
public class LYGSort{
    public List<Double> array = new ArrayList<Double>();
    public void sort(double [] a) {
        array.add(a[0]);
        if(a[1] > a[0]) {
            array.add(1, a[1]);
        }else {
            array.add(0, a[1]);
        }
        if(a[2] < a[0]) {
            array.add(0, a[2]);
        }else if(a[2] > a[1]) {
            array.add(2, a[2]);
        }else {
            array.add(1, a[2]);
        }
        for(int i = 3; i < a.length; i++)
            { bsa(a[i], 0, array.size() -
            1);
        }
    }
    private void bsa(double a, int l, int r)
        { int m = (l + r) >> 1;
        if(m != l) {
            if(a <=
                array.get(m)){ bsa
                (a, l, m);
            }else {
                bsa(a, m, r);
            }
        } else if(m == l){
            if(a <=
                array.get(m)){ arr
                ay.add(l, a);
            }else if(a > array.get(m) && a <=
                array.get(r) ){ array.add(l+1, a);
            }else {
                array.add(l+2, a);
            }
        }
    }
    public static void main(String agrs[])
        { int c=99999;
        double [] a = new double[c];
        java.util.Random r=new java.util.Random();
        for(int i=0;i<c;i++) {
```

```
            a[i]=r.nextDouble();
        }
        LYGSort lyg = new LYGSort();
        //timeCheck t= new timeCheck();
        //  t.begin();
        lyg.sort(a);
        //  t.end();
    }
}
```

==========

#函数名：基偶排序算法
#函数思想作者：基偶排序原理
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
public class OrderEvenSort{
    public int[] sort(int [] array)
        { boolean sorted = false;
        while(!sorted){
            sorted=true;
            for(int i = 1; i < array.length-1; i += 2)
                { if(array[i] > array[i+1]) {
                    swap(array,i, i+1);
                    sorted = false;
                }
            }
            for(int i = 0; i < array.length-1; i +=
                2){ if(array[i] > array[i+1]) {
                    swap(array, i, i+1);
                    sorted = false;
                }
            }
        }
        return array;
    }
    private void swap(int[] array, int i, int j)
        { int temp=array[i];
        array[i]=array[j];
        array[j]=temp;
    }
}
```

==========

#函数名：8叉数排序算法
#函数思想作者：八叉树原理
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
```

```
public class OTreeSort{
    public Leaf root;
    public Leaf heap;
    int c;
    int a1[];
    public int[] sort(int [] a) {
        //make tree
        c=0;
        a1=new int[a.length];
        if(root == null){
            root=new Leaf();
            root.value[0]=a[0];
            root.has01=1;
        }
        for(int
            i=1;i<a.length;i++){ hea
            p=root; addleaf(a[i]);
        }
        check(root);
        return a1;
    }
    public Leaf root(int[] a)
        { c=0;
        a1=new int[a.length];
        if(root == null){
            root=new Leaf();
            root.value[0]=a[0];
            root.has01=1;
        }
        for(int i=1;i<a.length;i++){
            heap=root;
            addleaf(a[i]);
        }
        return root;
    }
    private void check(Leaf temp)
        { if(temp!=null){
            check(temp.01);
            if(temp.has01==1){
                a1[c]=temp.value[0];
                c+=1;
            }
            check(temp.02);
            if(temp.has02==1){
                a1[c]=temp.value[1];
                c+=1;
            }
```

```
            check(temp.03);
            if(temp.has03==1){
                a1[c]=temp.value[2];
                c+=1;
            }
            check(temp.04);
            if(temp.has04==1){
                a1[c]=temp.value[3];
                c+=1;
            }
            check(temp.05);
            if(temp.has05==1){
                a1[c]=temp.value[4];
                c+=1;
            }
            check(temp.06);
            if(temp.has06==1){
                a1[c]=temp.value[5];
                c+=1;
            }
            check(temp.07);
            if(temp.has07==1){
                a1[c]=temp.value[6];
                c+=1;
            }
            check(temp.08);
        }
    }
    private void addleaf(int i){
        int count=heap.has01+heap.has02+heap.has03+heap.has04+heap.has05+heap.has06+heap.has07;
        if(count<7){
            if(count==1){
                heap.has02=1;
            }
            if(count==2){
                heap.has03=1;
            }
            if(count==3){
                heap.has04=1;
            }
            if(count==4){
                heap.has05=1;
            }
            if(count==5){
                heap.has06=1;
            }
            if(count==6){
```

```
            heap.has07=1;
        }
        heap.value[count]=i;
        heap.value=new InsertionSort().sort(heap.value,count+1);
        return;
    }else{
        if(i<=heap.value[0]){
            if(heap.01==null){
                heap.01=new Leaf();
                heap.01.value[0]=i;
                heap.01.has01=1;
                return;
            }else{
                heap=heap.01;
                addleaf(i);
            }
        }
        else if(i>heap.value[0] &&
            i<=heap.value[1]){ if(heap.02==null){
                heap.02=new Leaf();
                heap.02.value[0]=i;
                heap.02.has01=1;
                return;
            }else{
                heap=heap.02;
                addleaf(i);
            }
        }
        else if(i>heap.value[1] &&
            i<=heap.value[2]){ if(heap.03==null){
                heap.03=new Leaf();
                heap.03.value[0]=i;
                heap.03.has01=1;
                return;
            }else{
                heap=heap.03;
                addleaf(i);
            }
        }
        else if(i>heap.value[2] &&
            i<=heap.value[3]){ if(heap.04==null){
                heap.04=new Leaf();
                heap.04.value[0]=i;
                heap.04.has01=1;
                return;
            }else{
                heap=heap.04;
```

```
            addleaf(i);
        }
    }
    else if(i>heap.value[3] &&
        i<=heap.value[4]){ if(heap.05==null){
            heap.05=new Leaf();
            heap.05.value[0]=i;
            heap.05.has01=1;
            return;
        }else{
            heap=heap.05;
            addleaf(i);
        }
    }
    else if(i>heap.value[4] &&
        i<=heap.value[5]){ if(heap.06==null){
            heap.06=new Leaf();
            heap.06.value[0]=i;
            heap.06.has01=1;
            return;
        }else{
            heap=heap.06;
            addleaf(i);
        }
    }
    else if(i>heap.value[5] &&
        i<=heap.value[6]){ if(heap.07==null){
            heap.07=new Leaf();
            heap.07.value[0]=i;
            heap.07.has01=1;
            return;
        }else{
            heap=heap.07;
            addleaf(i);
        }
    }
    else{
        if(heap.08==null){ heap.
            08=new Leaf();
            heap.08.value[0]=i;
            heap.08.has01=1;
            return;
        }else{
            heap=heap.08;
            addleaf(i);
        }
    }
```

```
        }
    }
}
==========================================
#函数名：算法导论5代优化算法
#函数思想作者：算法导论快速排序算法4代基础。
#函数优化作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型
#函数解析：算法导论快速排序4代算法优化增加罗瑶光小高峰左右比对
package sortProcessor;
public class
    Quick_5D_Sort{ public int[]
    sort(int [] a) {
        quick2d(a,0,a.length-1);
        return a;
    }
    private void quick2d(int[] a, int lp, int rp)
        { int pos[]= new int[1];
        if(lp< rp)
        {partition(a,lp,rp,pos);
        quick2d(a,lp,pos[0]- 1);
        quick2d(a,pos[0]+ 1,rp);}
    }
    @SuppressWarnings("unused")
    private void partition(int[] a, int lp, int rp, int[]pos)
        { int x,lp1,rp1,temp;
        x= a[lp];rp1= rp;lp1= lp;
        int m= (rp+ lp)/ 2;
        int y= a[rp];
        if(x< y){//YAOGUANGLUO THEROY
            while(lp1< rp1){
                while((a[lp1]<= x)&& (lp1< rp1)) lp1 ++;
                while(a[rp1]>x)rp1 --;
                if(lp1<rp1){
                    temp=a[rp1];
                    a[rp1]=a[lp1];
                    a[lp1]=temp;
                }
            }
            a[lp]=a[rp1];
            a[rp1]=x;
        }else{//YAOGUANGLUO THEROY
            while(lp1<rp1){ while((a[lp1]<=y)&&(lp1<
                rp1)) lp1++; while(a[rp1]>y)rp1--;
                if(lp1<rp1){
                    temp=a[rp1];
```

```
                    a[rp1]=a[lp1];
                    a[lp1]=temp;
                }
            }
            a[lp]=a[rp1];
            a[rp1]=y;
        }
        pos[0]=rp1;
    }
}
```

==========

#函数名：罗瑶光快速排序6代算法

#函数优化作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型

#函数解析：基于罗瑶光快速排序5代算法优化增加递归深度优化并减少计算算子

```
package sortProcessor;
import timeProcessor.TimeCheck;
//第二代罗瑶光小高峰平均高峰过滤快排思想设计中。小高峰高峰过滤快速排序
public class Quick 6D luoyaoguang Sort{
    public int[] sort(int[] a) {
        TimeCheck imeCheck= new TimeCheck();
        imeCheck.begin();
        quick2ds(a, 0, a.length-1);
        imeCheck.end();
        imeCheck.duration();
        return a;
    }
    private void quick2ds(int[] a, int lp, int rp)
        { if(lp < rp){
            int c = rp - lp; if(c < 7){ int j;
            for(int i = 1 + lp; i <= lp + c; i++){
                j =
                    i;while(j>=1+lp)
                    { if(a[j]<a[j-
                    1]){
                        int temp=a[j];a[j]=a[j-1];a[j-1]=temp;
                    }
                    j--;
                }
            }
            return;
            }
            int pos = partition(a, lp, rp);
            quick2ds(a,lp,pos-1);
            quick2ds(a,pos+1,rp);
        }
```

```
}
private int partition(int[] a, int lp, int rp)
    { int x=a[lp];int lp1=lp;
     if(x>=a[rp]){
         x=a[rp];
    }
    while(lp1<rp){
        while(a[lp1]<=x &&lp1<rp )
            { lp1++;
        }
        while(a[rp]>x){
             rp--;
        }
        if(lp1<rp){
            int temp=a[rp];
            a[rp]=a[lp1];a[lp1]=temp;
        }
    }
    a[lp]=a[rp];a[rp]=x;
    return rp;
}
public String[][] sort(String[][] a)
    { quick2dsString(a, 0, a.length-1);
    return a;
}
private void quick2dsString(String[][] a, int lp, int rp)
    { if(lp < rp){
        int c = rp - lp; if(c < 7){ int j;
        for(int i = 1 + lp; i <= lp + c; i++){
            j =
                i;while(j>=1+lp){ if(Double.valueOf(a[j][1])<Double.va
                lueOf(a[j-1][1])){
                    String []temp=a[j];
                    a[j]=a[j-1];
                    a[j-1]=temp;
                }
                j--;
            }
        }
        return;
        }
        int pos = partitionString(a, lp, rp);
        quick2dsString(a,lp,pos-1);
        quick2dsString(a,pos+1,rp);
    }
}
private int partitionString(String[][] a, int lp, int rp)
```

```
        { String[] x=a[lp];int rp1=rp;int lp1=lp;
        if(Double.valueOf(x[1])>=Double.valueOf(a[rp][1])){
            x=a[rp];
        }
        while(lp1<rp1){ while((Double.valueOf(a[lp1][1])<=Double.valueO
    f(x[1]))&&(lp1<rp1)){
                lp1++;
            }
            while(Double.valueOf(a[rp1][1])>Double.valueOf(x[1])){   rp1--;
            }
            if(lp1<rp1){
                String[] temp=a[rp1];
                a[rp1]=a[lp1];a[lp1]=temp;
            }
        }
        a[lp]=a[rp1];a[rp1]=x;
        return rp1;
    }
}
================================================
#函数名：罗瑶光快速排序4代算法#函数优化作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型
#函数解析：基于罗瑶光快速排序6代算法优化增加递归深度优化，同频函数减少，同频算子减少，同频变量减少
package sortProcessor;
import timeProcessor.TimeCheck;
//第四代罗瑶光小高峰平均高峰过滤快排思想设计中。小高峰高峰过滤快速排序
//同频函数减少
//同频算子减少
//同频变量减少
public class
    Quick_Luoyaoguang_4D{ public
    int[] sort(int[] a) {
        TimeCheck imeCheck= new TimeCheck();
        imeCheck.begin();
        quick2ds(a, 0, a.length-1);
        imeCheck.end();
        imeCheck.duration();
        return a;
    }
    private void quick2ds(int[] a, int lp, int rp)
        { if(lp < rp){
            int c = rp - lp; if(c < 7){ int j;
            for(int i = 1 + lp; i <= lp + c; i++){
                j =
                    i;while(j>=1+lp)
                    { if(a[j]<a[j-
```

```
                    1]){
                        int temp=a[j];a[j]=a[j-1];a[j-1]=temp;
                    }
                    j--;
                }
            }
            return;
            }
            int pos = partition(a, lp, rp);
            quick2ds(a, lp, pos-1);
            quick2ds(a, pos+1, rp);
        }
    }
    private int partition(int[] a, int lp, int rp)
        { int x= a[lp]< a[rp]? a[lp]: a[rp];
        int lp1= lp;
        while(lp1< rp){//我总觉得这里可以进行一种积分算法优化，我一直在思考，别让那么快想到。
//          while(a[lp1]<= x&& lp1< rp) {
//              lp1++;
//          }
            while(!(a[lp1]>x|| lp1>=rp))
                { lp1++;
            } //今天想到了一些优化，
            while(a[rp]>x){
                rp--;
            }
            if(lp1<rp){
                int temp=a[rp];a[rp]=a[lp1];a[lp1]=temp;
            }
        }
        a[lp]=a[rp];a[rp]=x;
        return rp;
    }
    public String[][] sort(String[][] a)
        { quick2dsString(a, 0, a.length-1);
        return a;
    }
    private void quick2dsString(String[][] a, int lp, int rp)
        { if(lp< rp){
            int c= rp- lp; if(c< 7){ int j;
            for(int i= 1+ lp; i<= lp+ c; i++){
                j= i;while(j>= 1+ lp){
                    if(Double.valueOf(a[j][1])< Double.valueOf(a[j-
                        1][1])){ String []temp= a[j];
                        a[j]= a[j-1];
                        a[j-1]= temp;
                    }
```

```
                    j--;
                }
            }
            return;
            }
            int pos = partitionString(a, lp, rp);
            quick2dsString(a,lp,pos-1);
            quick2dsString(a,pos+1,rp);
        }
    }
    private int partitionString(String[][] a, int lp, int rp)
        { String[] x=a[lp]; int rp1=rp; int lp1=lp;
        if(Double.valueOf(x[1])>= Double.valueOf(a[rp][1])){
            x= a[rp];
        }
        while(lp1<
            rp1){ while((Double.valueOf(a[lp1][1])<=Double.valueOf(x[1]))&&(lp1<
            rp1)){
                lp1++;
            }
            while(Double.valueOf(a[rp1][1])>Double.valueOf(x[1])){    rp1--;
            }
            if(lp1<rp1){
                String[] temp= a[rp1];
                a[rp1]= a[lp1];a[lp1]= temp;
            }
        }
        a[lp]= a[rp1];a[rp1]= x;
        return rp1;
    }
}
```

==========

#函数名：选择排序

#函数思想作者：无

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型

```
package sortProcessor;
public class SelectionSort{
    public int[] sort(int [] array)
        { for(int i= 0; i< array.length;
        i++){
            for(int j= i+ 1; j< array.length;
                j++){ if(array[i]> array[j]){
                    int temp= array[i];
                    array[i]= array[j];
                    array[j]= temp;
                }
```

```
            }
        }
        return array;
    }
}
```

==========

#函数名：三叉排序
#函数思想作者：无
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型

```
package sortProcessor;
public class TTreeSort{
    public Leaf root;
    public Leaf heap;
    int c;
    int a1[];
    public int[] sort(int [] a) {
        //make tree
        c= 0;
        a1= new int[a.length];
        if(root== null){
            root= new Leaf();
            root.value[0]= a[0];
            root.has01= 1;
        }
        for(int i= 1; i< a.length;
            i++){ heap= root;
            addleaf(a[i]);
        }
        check(root);
        return a1;
    }
    public Leaf root(int[] a)
        { if(root== null){
            root= new Leaf();
            root.value[0]= a[0];
            root.has01= 1;
        }
        for(int i= 1; i< a.length;
            i++){ heap= root;
            addleaf(a[i]);
        }
        return root;
    }
    private void check(Leaf temp)
        { if(temp!= null){
            check(temp.01);
```

```
        if(temp.has01== 1){
            a1[c]=temp.value[0];
            c+= 1;
        }
        check(temp.02);
        if(temp.has02== 1){
            a1[c]= temp.value[1];
            c+= 1;
        }
        check(temp.03);
    }
}
private void addleaf(int i)
    { if(heap.has01== 1&& heap.has02==
    0){
        if(heap.value[0]<=
            i){ heap.value[1]=
            i; heap.has02= 1;
        }else{
            heap.value[1]= heap.value[0];
            heap.value[0]= i;
            heap.has02= 1;
        }
        return;
    }else{
        if(i<=
            heap.value[0]){ if(
            heap.01== null){
                heap.01= new Leaf();
                heap= heap.01;
                heap.value[0]= i;
                heap.has01= 1;
                return;
            }else{
                heap= heap.01;
                addleaf(i);
            }
        }
        else if(i> heap.value[0]&& i<=
            heap.value[1]){ if(heap.02== null){
                heap.02= new Leaf();
                heap= heap.02;
                heap.value[0]= i;
                heap.has01= 1;
                return;
            }else{
                heap= heap.02;
```

```
                    addleaf(i);
                }
            } else{
                if(heap.03==
                    null){ heap.03= new
                    Leaf(); heap=
                    heap.03;
                    heap.value[0]= i;
                    heap.has01= 1;
                    return;
                }else{
                    heap= heap.03;
                    addleaf(i);
                }
            }
        }
    }
}
```

==========

#函数名：线性数据叠加
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于加密和字符串计算

```
package steganographyProcessor;
public class LineCodeOperation{
    public String LineCodeADD(String lineCodeBoat, String lineCodeSet, int scale)
        { StringBuilder stringBuilder= new StringBuilder();
        for(int i= 0; i< lineCodeSet.length(); i+= scale) {
            int valueOfLineCodeBoat= Integer.valueOf(lineCodeBoat.charAt(i)).intValue();
            int valueOfLineCodeSet= Integer.valueOf(lineCodeSet.charAt(i)).intValue();
            int sum= valueOfLineCodeBoat+ valueOfLineCodeSet;
            stringBuilder.append(""+sum);
        }
        return stringBuilder.toString();
    }
}
```

==========

#函数名：线性数据与矩阵数据变换
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于加密和字符串计算

```
package steganographyProcessor;
public class PixFloat{
    public void floatPix(int[][] pix, int floatPix, int scale)
        { for(int x= 0; x< pix[0].length; x+= scale) {
            for(int y= 0; y< pix.length;y+= scale)
                { pix[x][y]+= floatPix;
```

```
            }
        }
    }
    public void arrangePix(int[][] pix, int arrangePix, int scale)
        { for(int x= 0; x< pix[0].length; x+= scale) {
            for(int y= 0;y< pix.length;y+= scale)
                { pix[x][y]+= arrangePix;
                pix[x+ 1][y]+= arrangePix-1;
                pix[x- 1][y]+= arrangePix-1;
                pix[x][y+ 1]+= arrangePix-1;
                pix[x][y- 1]+= arrangePix-1;
            }
        }
    }
    public int[] matrixToLiner(int[][] pix) {
        int[] output= new int[pix.length* pix[0].length];
        for(int x= 0; x< pix[0].length; x++)
            { for(int y= 0; y< pix.length; y++)
            {
                output[x* pix[0].length+ y]= pix[x][y];
            }
        }
        return output;
    }
    public String linerToLineCode(int[] pix)
        { StringBuilder code= new
        StringBuilder(); Here:
            for(int i= 0; i< pix.length; i++){
                String register = String.valueOf(pix[i]);
                if(register.length()== 3) {
                    code.append(""+ register);
                    continue Here;
                }
                if(register.length()== 2)
                    { code.append("0"+
                    register); continue Here;
                }
                if(register.length()== 1)
                    { code.append("00"+
                    register);
                }
            }
        return code.toString();
    }
}
```

==========

```
package waveProcessor;
```

```
public class Common{
    public double[] zhiShu(double[] input, double scale,double shehold)
        { double [] output= new double [input.length];
        for(int i=0; i<input.length;i++)
            { output[i]= input[i];
            if(output[i]< shehold)
                for(int j= 0; j<scale- 1; j++)
                    { output[i]*= output[i];
                }
        }
        return output;
    }
}
```

==========

#函数名：波数据复制

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型，用于加密和字符串计算

```
package waveProcessor;
public class Copy{
    public double[][] copy2d(double[][] input, double scale)
        { double[][] output= new
        double[(int)scale][input[0].length]; for(int i=0; i< scale;
        i++) {
            for(int j=0; j< input[0].length;j++)
                { output[i][j]= input[i][j];
            }
        }
        return output;
    };
    public double[]copy1d(double[] input, double scale)
        { double[] output= new double[(int)scale];
        for(int i=0; i<scale; i++)
            { output[i]= input[i];
        }
        return output;
    };
    public double[]copy1dx2(double[] input,double scale)
        { double[] output= new double[(int)(scale*
        input.length)]; for(int i=0; i<scale; i++) {
            for(int j=0; j<input.length; j++)
                { output[i* input.length+ j]=
                input[j];
            }
        }
        return output;
    }
```

```
}
```

==========

#函数名：波数据傅里叶变换
#函数思想作者：傅里叶
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于频率变换

```
package waveProcessor;
public class DFT{
    double Pi=3.1415926;
    public double [] ft(double input[]) {
        double output[]= new double[input.length];
        double cos[][]= this.ft_cos(input.length);
        double sin[][]= this.ft_sin(input.length);
        for(int k=0; k<input.length; k++){
            double r= 0, i= 0;
            for(int   n=0;   n<   input.length;
                n++){ r+= input[n]* cos[k][n];
                i+= input[n]* sin[k][n];
            }
            output[k]= Math.sqrt(r* r+ i* i);
        }
        return output;
    }
    public double [][] ft_cos(double size) {
        double cos[][]= new double[(int)size][(int)size];
        for(int k=0; k<(int)size; k++){
            for(int n=0; n<(int)size; n++){
                cos[k][n]= Math.cos(2* Pi* k* n/ (int)size);
            }
        }
        return cos;
    }
    public double [][] ft_sin(double size) {
        double sin[][]= new double[(int)size][(int)size];
        for(int k=0; k<(int)size; k++){
            for(int n=0; n<(int)size; n++){
                sin[k][n]= Math.sin(2* Pi* k* n / (int)size);
            }
        }
        return sin;
    }
}
```

==========

#函数名：波数据拉普拉斯变换
#函数思想作者：拉普拉斯
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于曲度变换

```
package waveProcessor;
public class
    Laplasian{ @SuppressWarnings("un
    used")
    public double[] laplasian1d(double[] input, double scale )
        { double [] output= new double [input.length];
        double [] lap= new double[5];
        double sig = scale; //default 1.6
        double t = 0;
        double sumhere = 0;
        lap[0]= 0;
        lap[1]= -3;
        lap[2]= scale;//default=7
        lap[3]= -3;
        lap[4]= 0;
        for(int l= 0; l< 5;
            ++l){ sumhere= sumhere+
            lap[l];
        }
        double sum1= 0;
        for(int j=0; j<5;
            ++j){ lap[j]= lap[j]/
            sumhere; sum1= sum1+
            lap[j];
        }
        double sum=0;
        for(int i=2;i<input.length-
            2;i++){ sum=0;
            for(int j= -2; j< 3; j++){
                sum = sum + (input[i+ j]* lap[j+ 2]);
            }
            output[i]=sum;
        }
        return output;
    }
}
```

==========

#函数名：波数据波峰波谷计算

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型，用于波统计

```
package waveProcessor;
public class MaxAndMin{
    public double max_v(double[] input)
        { double max = 0;
        for(int i=0;i<input.length;i++)
            { if(input[i]>max) {
```

```
                max=input[i];
            }
        }
        return max;
    }
    public double max_i(double[] input)
        { double max = 0;
        for(int i=0;i<input.length;i++)
            { if(input[i]>max) {
                max=i;
            }
        }
        return max;
    }
    @SuppressWarnings("unused")
    public double min_v(double[] input,double rank)
        { double min = 999999999;
        double[][] fengtong =new waveProcessor.PeakStatistic().fengTong1(input);
        for(int i=0; i<input.length; i++) {
            if(input[i] < min)
                { min=input[i];
            }
        }
        return min;
    };
    public double min_i(double[] input, double rank)
        { double min = 999999999;
        for(int i=0; i<input.length; i++)
            { if(input[i] < min) {
                min=i;
            }
        }
        return min;
    }
}
```

==========

#函数名：波数据平滑均值算法

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型，用于波平滑过滤

```
package waveProcessor;
public class Median{
    public double[] median1d(double[] input, double scale )
        { double [] output= new double [input.length];
        for(int i= (int)scale;i < input.length- scale; i++)
            { double sum= input[i];
            for(int j= 1;j< scale; j++)
```

```
                { sum+= input[i + j];
                sum+= input[i - j];
            }
            sum/= scale* 2+ 1;
            output[i]= sum;      ;
        }
        return output;
    }
}
```

==========

#函数名：波峰统计函数
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于波的峰统计和地址位排序。

```
package waveProcessor;
public class PeakStatistic{
    public double[][] fengTong1(double[] input)
        { double[][] output= new
        double[input.length][2]; for(int i= 0; i<
        input.length; i++) {
            output[i][0]= i;
            output[i][1]= input[i];
        }
        return output;
    }
    public double[][] fengPaixX(double[][] input) {
        double[][] output= new waveProcessor.Copy().copy2d(input,input.length);
        for(int i= 0; i< input.length; i++) {
            for(int j= 0; j< input.length; j++)
                { if(output[i][0]<output[j][0]) {
                    double tempc[]= new double [2];
                    tempc[0]= output[i][0];
                    tempc[1]= output[i][1];
                    output[i][0]= output[j][0];
                    output[i][1]= output[j][1];
                    output[j][0]= tempc[0];
                    output[j][1]= tempc[1];
                }
            }
        }
        return output;
    }
    public double[][] fengPaiyY(double[][] input) {
        double[][] output =new waveProcessor.Copy().copy2d(input,input.length);
        for(int i= 0; i< input.length; i++) {
            for(int j= 0; j< input.length; j++)
                { if(output[i][1]< output[j][1])
```

```
            {
                double tempc[]= new double [2];
                tempc[0]= output[i][0];
                tempc[1]= output[i][1];
                output[i][0]= output[j][0];
                output[i][1]= output[j][1];
                output[j][0]= tempc[0];
                output[j][1]= tempc[1];
            }
        }
    }
    return output;
}
public double[][] fengPaiXx(double[][] input) {
    double[][] output= new waveProcessor.Copy().copy2d(input,input.length);
    for(int i= 0; i < input.length; i++) {
        for(int j= 0; j< input.length; j++)
            { if(output[i][0]> output[j][0])
            {
                double tempc[]= new double [2];
                tempc[0]=output[i][0];
                tempc[1]=output[i][1];
                output[i][0]=output[j][0];
                output[i][1]=output[j][1];
                output[j][0]=tempc[0];
                output[j][1]=tempc[1];
            }
        }
    }
    return output;
}
public double[][] fengPaiYy(double[][] input) {
    double[][] output =new waveProcessor.Copy().copy2d(input,input.length);
    for(int i = 0; i < input.length; i++) {
        for(int j = 0; j < input.length; j++)
            { if(output[i][1]>output[j][1]) {
                double tempc[]=new double [2];
                tempc[0]=output[i][0];
                tempc[1]=output[i][1];
                output[i][0]=output[j][0];
                output[i][1]=output[j][1];
                output[j][0]=tempc[0];
                output[j][1]=tempc[1];
            }
        }
    }
    return output;
```

```
    }
}
```

==========

#函数名：波数据比例重洗

#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型，用于波的比例缩放，用于德塔养疗经，华瑞集的语音处理。

```
package waveProcessor;
public class Proportion{
    public double[] newX(double[] input, double width) {//128 32
        double [] output= new double [(int)width];
        double bilix= input.length/ width;//4
         for(int i=0; i<output.length; i++)
           { for(int j=0; j<bilix; j++) {
                output[i]+= input[(int)(i*bilix+j)];
            }
        }
        return output;
    };
    public double[] newY(double[] input, double hight)
        { double [] output= new double [input.length];
        double max = new waveProcessor.MaxAndMin().max_v(input);
        double biliy=  hight/max;
        for(int i=0; i<output.length;i++)
            { output[i]= input[i]*biliy;
        }
        return output;
    }
    public double[] newXY(double[] input, double width, double hight )
        { double [] output= new double [(int)width];
        double max = new waveProcessor.MaxAndMin().max_v(input);
        double biliy=  hight/ max;
        double bilix=  input.length/ width;
        for(int i=0; i<output.length; i++) {
            output[i]= input[(int)(i* bilix)]* biliy;
        }
        return output;
    }
    public double[] newXYBest(double[] input, double width, double hight )
        { double [] output= new double [(int)width];
        double max = new waveProcessor.MaxAndMin().max_v(input);
        double biliy=  hight/ max;
        double bilix=  width/ input.length;
        for(int i=0; i<input.length-1; i++) {
            double dc= (input[i+1]-input[i])/bilix;
            for(int j=0; j<bilix; j++) {
                output[(int)(i* bilix)+ j]= (input[i]+ dc* j)* biliy;
```

```
            }
        }
        return output;
    };
    public double[] newYwithoutBound(double[] input, double hight)
        { double [] output= new double [input.length];
        for(int i= 0; i< output.length; i++)
            { output[i]= input[i]* hight;
        }
        return output;
    };
    public double[] newXYYwithoutBound(double[] input, double width, double hight )
        { double [] output= new double [(int)width];
        double bilix=  input.length/width;
        for(int i= 0; i< output.length; i++) {
            output[i]= input[(int)(i* bilix)]* hight;
        }
        return output;
    };
}
```

==========

#函数名：波数据量化重洗
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于波的比例缩放，作者针对股市数据设计的函数。

```
package waveProcessor;
public class Quantification{
    public double[] liangHuaDengChaAdd(double[] input, double scale)
        { double[] output= new double[input.length];
        double sum= input.length/ scale;
        for(int i=0; i<sum-1; i++) {
            double temp=0;
            for(int j=0;j<scale;j++)
                { temp+=input[(int)(i*scale+j)]
                ;
            }
            output[(int)(i*scale)]=temp;
        }
        return output;
    }
    public double[] liangHuaXiHua(double[] input,double scale)
        { double[] output= new double[input.length];
        double sum= input.length;
        int find= 0;
        for(int i=1; i<=scale; i++)
            { if(input[0]> input[i])
            {
```

```
                find+= 1;
            }
        }
        if(find== scale)
            { output[0]=
            input[0];
        }
        for(int i= (int)(scale); i<(sum- (scale)); i++)
            { find= 0;
            for(int j= 1; j<= scale; j++)
                { if(input[i]> input[i+ j])
                {
                    find+= 1;
                }
                if(input[i]> input[i- j])
                    { find+= 1;
                }
            }
            if(find==scale* 2)
                { output[i]=input[i]
                ;
            }
        }
        return output;
    }
    @SuppressWarnings("unused")
    public double[] liangHuaDengChaMines(double[] input, int scale)
        { double[] output= new double[input.length];
        double sum= input.length/ scale;
        for(int i= 0;i< sum- 1; i++) {
            double temp= 0;
            double max= input[(int)(i*scale)];
            double maxi= i* scale;
            for(int j= 0; j< scale; j++)
                { if(input[(int)(i* scale+ j)]> max)
                {
                    max= input[(int)(i* scale+ j)];
                    maxi= (int)(i* scale+ j);
                }
            }
            output[(int)(i*scale)] = max;
        }
        return output;
    }
    public double[] liangHuaEqualDelete(double[] input)
        { double[] output = new double[input.length];
        double pre= 0;
```

```
        double next= 0;
        for(int i=0; i<input.length; i++)
            { next= input[i];
            if(next!= pre)
                { output[i]=
                input[i];
            }else {
                output[i]= 0;
            }
            pre= next;
        }
        return output;
    }
    public double[] liangHuaXiHuaHalfSide(double[] input)
        { double[] output= new double[input.length];
        for(int i=1; i< input.length-1; i++){
            if(input[i]/input[i+1]<= 1&& input[i]/ input[i+1]>= 0.5)
                { if(input[i]/ input[i-1]<= 1&& input[i]/ input[i-1]>=0.5)
                {
                    output[i]= input[i];
                }
            }
        }
        return output;
    }
}
================================================
```

#函数名：波数据极化分离
#函数思想作者：罗瑶光
#函数功能作者：罗瑶光
#函数用途：自己写的验证原型，用于波数据的归极分化。

```
package waveProcessor;
public class Shehold{
    public double[] shehold(double[] input, double scale)
        { double [] output= new double [input.length];
        for(int i=0;i<input.length;i++) {
            if(input[i]> scale)
                { output[i]=100;
            }
        }
        return output;
    }
}
================================================
```

#函数名：波数据裁剪函数
#函数思想作者：罗瑶光

#函数功能作者：罗瑶光

#函数用途：自己写的验证原型，用于波的裁剪提取。用于德塔养疗经，华瑞集的语音处理

```
package waveProcessor;
public class Tailor{
    public double[] caiJian(double[] input, double left, double right)
        { double w= (int)(right- left);
        double [] output= new double [(int)w];
        for(int i=(int)left;i<right;i++)
            { output[(int)(i-left)]=input[i];
        }
        return output;
    };
}
```