## 数据结构实验——图片网络爬虫设计与图像处理

161220017陈翔

# 总体设计框架

main函数

通过布隆过滤器

Python

GotoSites函数(搜索网站)

GetPictures函数(下载图片)

使用Mat类型读取图片

用PixImage类读取Mat类型

C++

用PixImage类读取Mat类型

使用PixImage类中的bluring函数对图像进行模糊化处理

使用PixImage类中的sobel函数对图像进行边缘检测

# 实现细节

## python部分

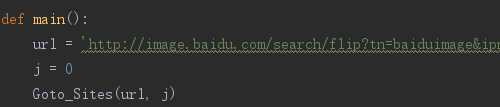
### 2.1.1概述

作者使用的python版本为3.6，显式调用的python库有两个，分别为re和requests。

在本实验中，作者一共定义了三个全局函数和一个类，三个函数名分别为main，Goto\_Sites和Get\_Picture，定义的类名为Bfilter。下面具体阐述它们的功能。

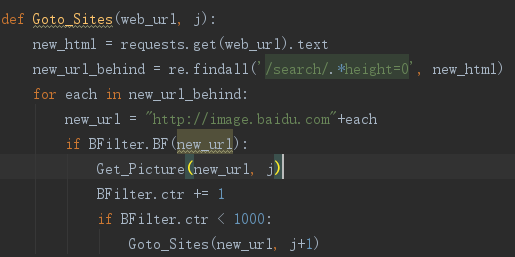
### 2.1.2main

该函数为主函数，程序从该函数开始执行。函数对爬取的初始网站进行了设置，然后以此为参数调用Goto\_Sites函数。



### 2.1.3Goto\_Sites

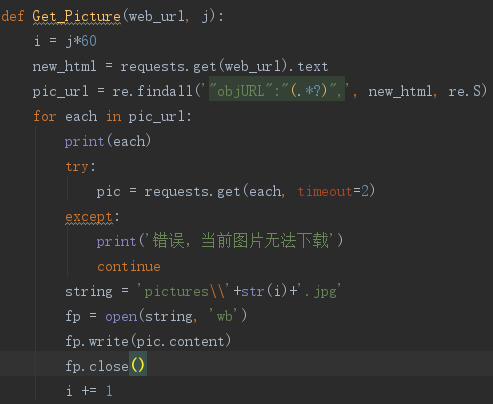
该函数将传入的参数中的文档内容进行了读取，然后搜索其中所有可能含有网页地址的部分（通过正则表达式），并将其中的地址读取之。对于每一个新地址，我们会首先对其是否已经访问过进行判断，方法是通过布隆过滤器，这个我们下面会详细讲述。如果通过布隆过滤器，说明该地址没有被访问过，则调用Get\_Picture函数，参数为当前地址，紧跟着继续调用（体现深度优先）Goto\_Sites函数，参数为当前地址。



注：这里的查找网站所使用的正则表达式是针对百度图片的特例。

### 2.1.4Get\_Picture

该函数将传入的参数中的文档内容进行了读取，然后搜索其中所有可能含有图片的部分，并将其中的图片地址打印之，接着对其进行下载操作。因为可能有无法下载的情况，为了避免程序进入假死状态，将下载语句放在try中，并规定如果响应时间超过2秒，则输出错误信息，继续下一张图片的下载。

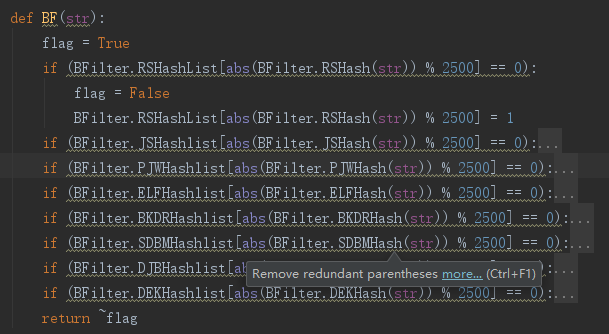


注：这里的查找图片所使用的正则表达式是针对百度图片的特例。

### 2.1.5BFilter

这是一个类，其中的成员包括8个hash函数，8个用来存放不同hash表的容量为2500bit的数组，以及一个判断字符串以在过滤器中的函数（这也是函数的接口）。八个hash函数已在讲义中给出，这里不重复。下面具体介绍一下程序接口的实现。

该函数的参数为一个字符串。在函数体中，首先声明一个值为false的布尔量flag，然后依次调用八个hash函数，如果有一个hash函数返回值为True，则说明该字符串不在原有的集合中，则将flag的值改为true然后该字符串的映射位置在对应的hash表中置一。最后返回值为非flag。



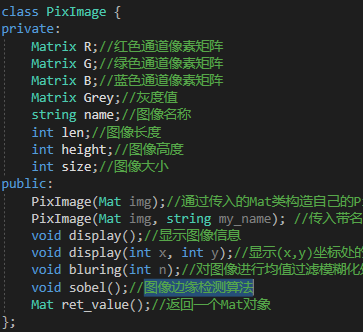
假设集合中已经存在1000个元素了，那么它们在任意hash函数的映射下的地址数一定小于等于1000。假设它们最大都可以映射到1000个地址，那么接下来第1001个元素在每个hash函数下映射到一个非空地址的平均概率为0.4，则八个hash函数都认为它已经被访问过的概率为1/(0.4)8=0.0655%，当然这还包括他的确已被访问过的概率，故假阳性概率远远小于0.1%。

## C++部分

### 2.2.1概述

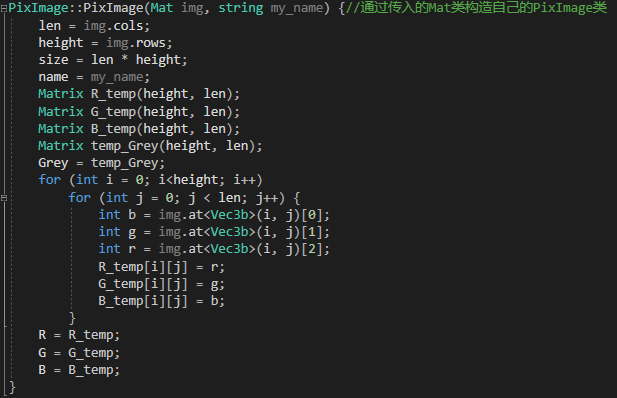
作者使用了opencv库用来读取图片格式和显示图片效果，其余操作都由自定义的PixImage类来完成。接下来我们介绍一下这个类。

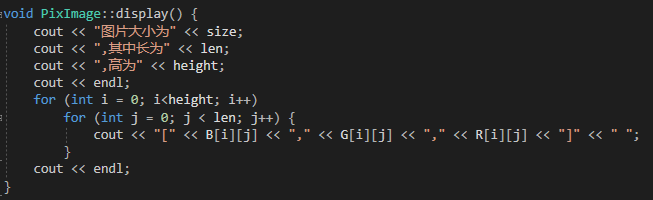
### 2.2.2PixImage

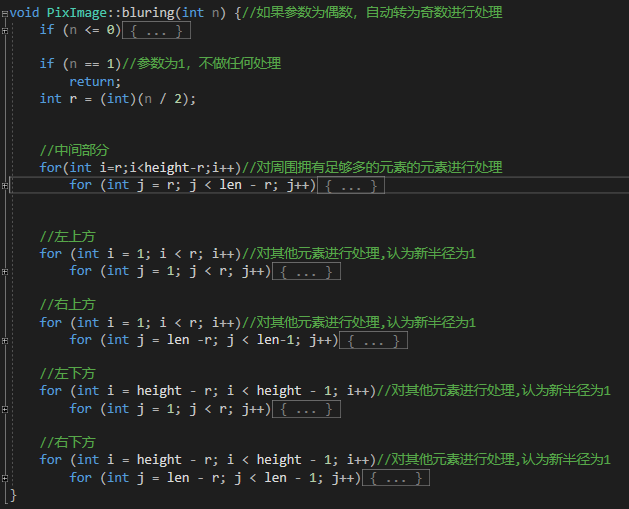


该类有八个成员变量和七个成员函数，其中R,G,B，Grey的类型为自定义的Matrix类，这是作者以前实现的，包括了一些矩阵常用的功能。七个成员函数中实际上只有5个不同名的函数。PixImage函数为构造函数，通过一个传入的Mat类型来初始化成员变量。display函数用来显示图像像素信息。bluring函数用来对图像进行均值过滤。sobel用来对图像进行边缘检测。ret\_value用来返回一个Mat对象。

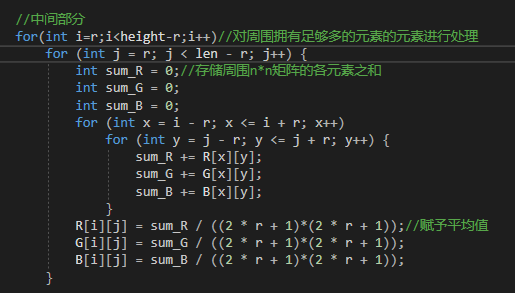
* PixImage():用来根据传入的Mat对象初始化各成员变量



* display():
* bluring(int n):均值过滤模糊化处理



其中中间部分的代码为为将某一元素周围半径r的元素平均值赋予该元素：



* sobel():用来进行边缘检测

void PixImage::sobel() {

Mat Greyimg; //声明一个灰度图

Mat img = ret\_value();//返回当前对象对应的Mat对象

cvtColor(img, Greyimg, CV\_BGR2GRAY); //将该Mat对象转为灰度图

imshow("Greyimage", Greyimg); //展示灰度图

for (int i = 0; i<height; i++)

for (int j = 0; j < len; j++) {

Grey[i][j] = Greyimg.at<uchar>(i, j);

}

Matrix Gx(height, len);

Matrix Gy(height, len);

Matrix G(height, len);

for (int i = 1; i<height - 1; i++)//对周围拥有足够多的元素的元素进行处理

for (int j = 1; j < len - 1; j++) {

Gx[i][j] = (-1)\*Grey[i - 1][j - 1] + 0 \* Grey[i][j - 1] + 1 \* Grey[i + 1][j - 1]

+ (-2)\*Grey[i- 1][j] + 0 \* Grey[i][j] + 2 \* Grey[i + 1][j]

+ (-1)\*Grey[i - 1][j + 1] + 0 \* Grey[i][j + 1] + 1 \* Grey[i + 1][j + 1];

Gy[i][j] = 1\*Grey[i - 1][j - 1] + 2\* Grey[i][j - 1] + 1 \* Grey[i + 1][j - 1]

+ 0\*Grey[i - 1][j] + 0 \* Grey[i][j] + 0 \* Grey[i + 1][j]

+ (-1)\*Grey[i - 1][j + 1] + (-2) \* Grey[i][j + 1] + (-1) \* Grey[i + 1][j + 1];

G[i][j] = (int)sqrt(pow(Gx[i][j], 2) + pow(Gy[i][j], 2));

//cout << G[i][j] << " ";

}

long long sum=0;

for (int i = 1; i<height - 1; i++)//对周围拥有不足量的元素的元素进行处理

for (int j = 1; j < len - 1; j++) {

sum += Grey[i][j];

}

int mean = (int)(sum / ((height - 2)\*(len - 2)));

int scale = 30; //可以修改的值

int cutoff = scale \* mean;

int thresh = (int)sqrt(cutoff);//阈值

//将灰度图转为黑白图

for (int i = 1; i<height - 1; i++)

for (int j = 1; j < len - 1; j++) {

if (G[i][j] > thresh)

Greyimg.at<uchar>(i, j) = 0xFF;

else

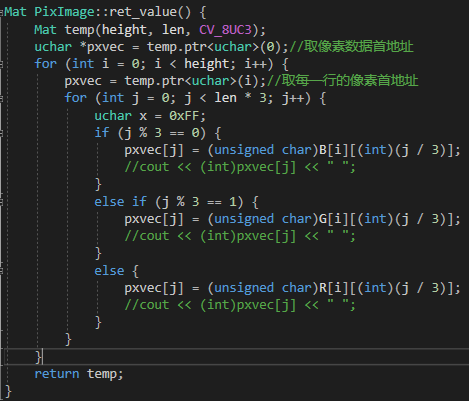
Greyimg.at<uchar>(i, j) = 0x00;

}

imshow("BAWimage", Greyimg);

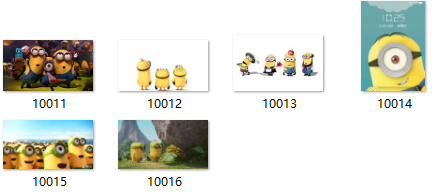
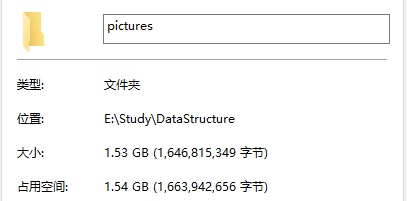
}

* ret\_value():用来返回一个Mat对象，方便显示操作

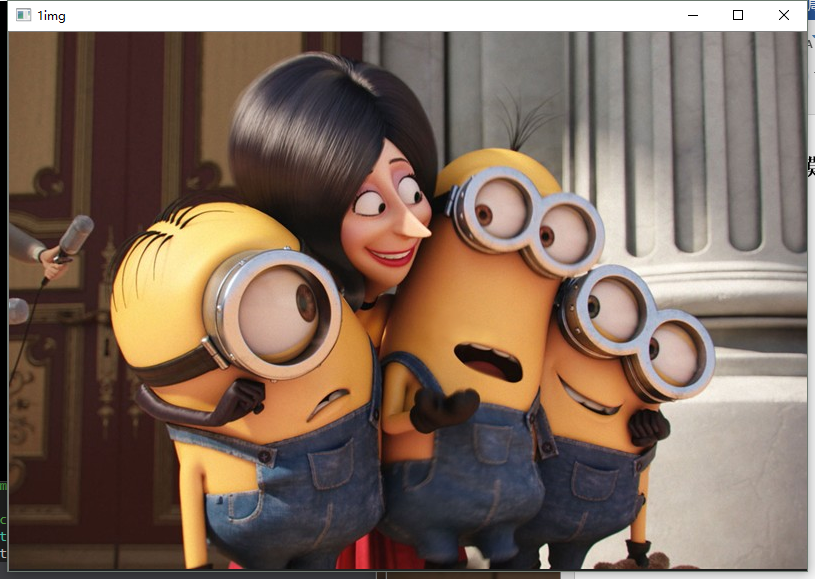
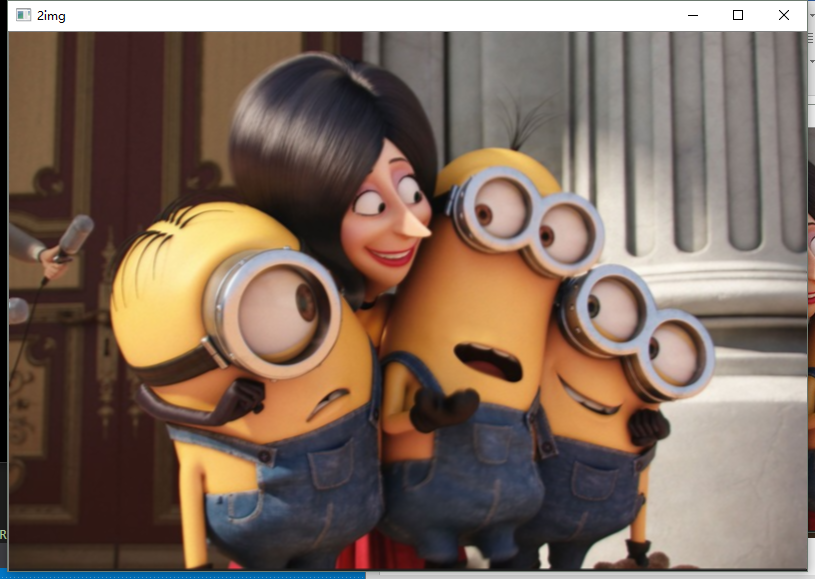
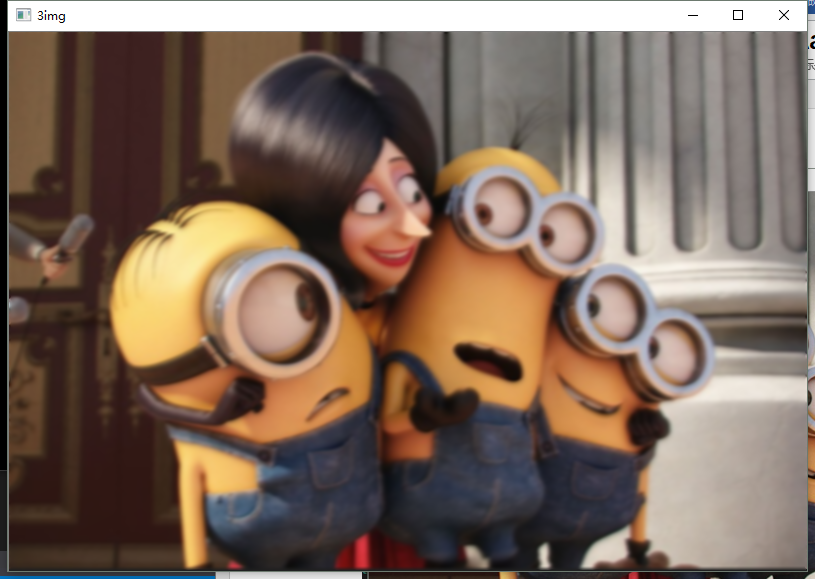
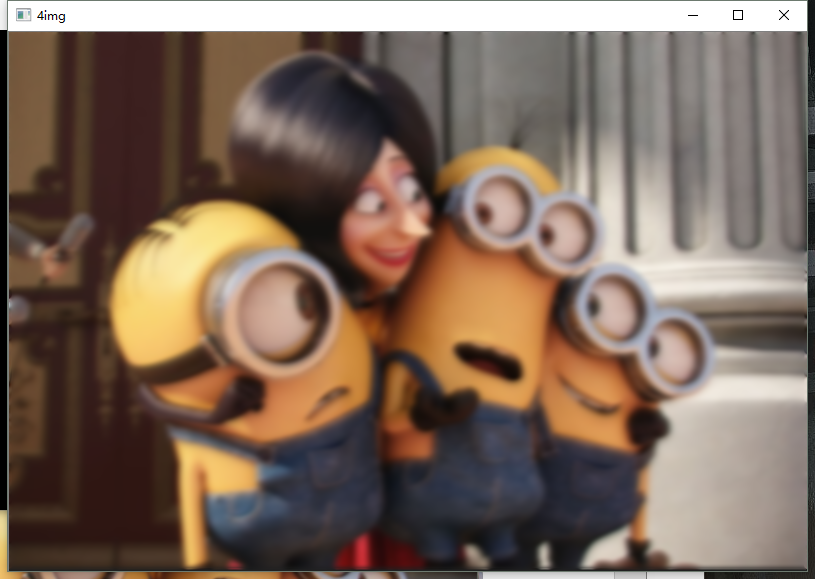


# 最终效果

## 爬到的图片：

## 图片模糊化：

## 图片边缘检测：

