

情報メディアプロジェクト I：演習課題レポート 2

澤 祐里 (21-1-037-0801)

2021 年 7 月 3 日

目次

1	目的	1
1.1	課題 1	1
1.2	課題 2	1
2	手順	2
2.1	課題 1	2
2.1.1	計測対象	2
2.1.2	計測方法	3
2.2	課題 2	6
2.2.1	計測対象	6
2.2.2	計測方法	6

1 目的

1.1 課題 1

プログラム課題から一つを選び、1 枚の画像を処理する時間を以下の点に気を付けながら計測する。

注意点 1 …計測対象は何かを示す。

注意点 2 …計測方法は正確かどうかを調べる。

注意点 3 …ループ回数を増減すると一回の処理当たりの所要時間が変化する理由を分析する。

1.2 課題 2

任意のプログラム課題において、同じ処理でもアルゴリズムを変更すると、`int` や `double` の計算を行う回数が増える。また、配列などのメモリに対するアクセス回数も変わり、オブジェクトの生成やメソッドの呼び出しに時間がかかることなどにも注意して処理を比較する。課題 2 においても、以下の点に注意しながら計測する。

注意点 1 …計測対象は何かを示す。

注意点 2 …計測方法は正確かどうかを調べる。

注意点 3 …アルゴリズムと処理時間の関係について分析する。

2 手順

2.1 課題 1

2.1.1 計測対象

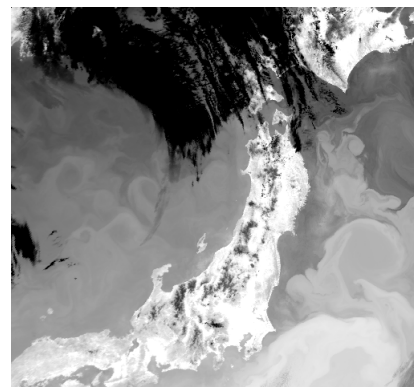
- 計測対象の選び方については、課題 1 だけを考えると任意のプログラムでよい。しかし、課題 2 でも課題 1 の計測結果が転用できることを考慮すると、処理結果が変わらないように、アルゴリズムだけを変更できるプログラムを選びたい。
 - 画像のノイズを除去することができる「 3×3 メディアンフィルタを使った平滑化」では、中央値を見つけるためにソートを行う。ソートに用いられるアルゴリズムは多数発見されているため、アルゴリズムを変更しやすい上に処理結果には影響しない。
 - 以上のことから、計測対象には、「 3×3 メディアンフィルタを使った平滑化」を選ぶことにする。具体的には、図 1 のようなプログラムを使う。
- 画像処理に用いる画像の選び方については、任意の画像でよい。しかし、計測時間があまりにも長くならないようにするため、大きな画像は選ばない。また、画像の大きさや画素値などが計測時間にも影響すると考えられるため、全ての計測で使う画像は、1 枚の同じ画像に統一して行う。
 - 今回の実験で使う画像は、大きさが「幅 512 ピクセル × 高さ 479 ピクセル」の図 2 のような画像「d850429avhrr4.bmp」である。

図 1 メディアンフィルタを使った平滑化のプログラム

```
private static GImage noisedelete(GImage inputimg,int range){ //rangeは原点からフィルタにする範囲
    int width = inputimg.getWidth();
    int height = inputimg.getHeight();
    GImage outputimg = new GImage(height,width); //出力用画像を生成
    int[] median = new int[(int)Math.pow(2*range+1 , 2)]; //入力された範囲からフィルタ用の配列を生成
    int count; //フィルタに画素値を挿入していくためのカウント用変数
    int exchange; //ソートで使う交換用変数

    for(int y=0;y<height;y++){
        for(int x=0;x<width;x++){
            if(y-range >= 0 && y+range < height && x-range >= 0 && x+range < width){
                count = 0;
                for(int i=-range;i <= range;i++){
                    for(int j=-range;j <= range;j++){
                        median[count] = inputimg.pixel[y + i][x + j];
                        count++;
                    }
                    if(count == median.length){
                        for(int a=0;a<median.length-1;a++){
                            for(int b=median.length-1;b>a;b--){
                                if(median[b] > median[b-1]){
                                    exchange = median[b];
                                    median[b] = median[b-1];
                                    median[b-1] = exchange;
                                }
                            }
                        }
                    }
                    outputimg.pixel[y][x] = median[(median.length-1)/2];
                }
            }
            else{
                outputimg.pixel[y][x] = GImage.MIN_GRAY;
            }
        }
    }
    return outputimg;
}
```

図 2 計測対象の画像処理に使う画像



2.1.2 計測方法

1. 計測方法の選択については、コストや精度の面から考えて、人力よりもコンピューター自体にやらせた方が効率がいいと判断したため、プログラミングで計測システムを作成する。

→ java では、「System.currentTimeMillis()」というメソッドを使うことで、long 型でエポック秒 (1970 年 1 月 1 日 0 時 0 分 0 秒) から経過した時間をミリ秒単位で知ることができる。この実験では、このメソッドを利用して画像処理の処理時間を計測していく。

→ 具体的な計測方法は「計測したい処理」を「System.currentTimeMillis()」で囲いこむことで、処理前の時間と処理後の時間を取得する。そして、(2.1.1) の式のように処理後の時間から処理前の時間を引くことで、処理時間を計算する。

$$\text{処理時間} = \text{処理後の時間} - \text{処理前の時間} \quad (2.1.1)$$

2. 次に、選択した計測方法が正確であるかどうかを判断できる条件を考える。

条件一 今回の実験では、画像処理の処理時間だけを計測したいため、それ以外の処理の計測時間は排除したい。私が選択した計測方法で、画像処理の処理時間以外に、計測時間に反映されと思われる処理は、計測に使うメソッドである「System.currentTimeMillis()」という処理自体が挙げられる。そのため、このメソッドに処理時間が存在するのかどうかを調べる。

条件二 本題では、正解の処理時間が不明であるため、正確に処理時間が計測できているかどうかを判断することができない。そのため、処理時間があらかじめ決められている処理を利用して、条件一の結果を含めた処理時間を正確に測れるかどうかをテストする。

条件三 条件一、条件二は、私のコンピューターでの時間の進み方に依存しており、それが現実の時間の進み方と同じであるという前提で成り立っている。そのため、私のコンピューター以外の方法でも測ることで、前提が正しいかを調べた上で、プログラムに対して私の解釈違いが起こっていないのかも調べることができる。

3. 2 で考えた条件を実際に確かめてみる。

- 条件一

方法 計測方法は、図 3 のように、「System.currentTimeMillis()」を long 型の変数「starttime」と「endtime」に入れる。その後、「endtime」から「starttime」を引いて処理時間を計算する。また、信頼性のあるデータをとるために、これらの処理を for 文で 20 回繰り返している。

結果 「System.currentTimeMillis()」の処理時間の計測結果は、図 4 のように、試行回数 20 回の全てで「0ms」となった。これらの結果より、「System.currentTimeMillis()」の処理時間はこの実験では考慮しなくてもよいと判断した。

図 3 「System.currentTimeMillis()」の処理時間を計測するプログラム

```
private static void milltime(){
    for(int i=0;i<20;i++){
        long starttime = System.currentTimeMillis();

        long endtime = System.currentTimeMillis();
        System.out.println("開始時刻:"+starttime+"ms");
        System.out.println("終了時刻:"+endtime+"ms");
        System.out.println("処理時間:"+endtime - starttime+"ms");
    }
}
```

図 4 「System.currentTimeMillis()」の処理時間の計測結果

```
開始時刻:1625128208561ms
終了時刻:1625128208561ms
処理時間:0ms
開始時刻:1625128208561ms
終了時刻:1625128208561ms
処理時間:0ms
開始時刻:1625128208561ms
終了時刻:1625128208561ms
処理時間:0ms
開始時刻:1625128208562ms
終了時刻:1625128208562ms
処理時間:0ms
```

- 条件二

方法 java の「Thread.sleep();」メソッドでは、引数に入れた数字×ミリ秒だけ、プログラムを一時停止することができます。これを利用して「Thread.sleep();」を処理とみなして、処理時間を計測し、「Thread.sleep();」の引数との比較を行うことで、計測方法が正確かどうかを判断する。この実験では、「Thread.sleep();」の引数を「1000」として、計測処理を for 文で 20 回繰り返す図 5 のようなプログラムを作成して計測した。

結果 結果は、図 6 のようになり、結果をまとめると「1001ms」が 9 回、他は全て「1000ms」となった。1ms の誤差が約 1/2 の確率で発生したものの、この程度の誤差はマシンの性能などによる誤差の範疇だと考察できるため、この計測方法は正確であると判断した。

図 5 「Thread.sleep();」を用いた計測方法の正確さ確認のプログラム

```
private static void Checkaccuracy(){
    for(int i=0;i<20;i++){
        long starttime = System.currentTimeMillis();
        try {
            Thread.sleep(1000);
        }
        catch (InterruptedException e) {
        }
        long endtime = System.currentTimeMillis();
        System.out.println("開始時刻:"+starttime+"ms");
        System.out.println("終了時刻:"+endtime+"ms");
        System.out.println("処理時間:"+ (endtime - starttime)+"ms");
    }
}
```

図 6 「Thread.sleep();」を用いた計測方法の正確さ確認の出力結果

```
開始時刻:1625135123951ms
終了時刻:1625135124951ms
処理時間:1000ms
開始時刻:1625135124951ms
終了時刻:1625135125952ms
処理時間:1001ms
開始時刻:1625135125952ms
終了時刻:1625135126952ms
処理時間:1000ms
```

- 条件三

方法 計測に用いるコンピュータの機能で計測しても意味がないため、スマートフォンのストップウォッチを用いて手動で測る。手動のため「Thread.sleep();」の引数を長めの「20000」に設定し、開始と終了のログがでるように、図 7 のようなプログラムを使う。

結果 ストップウォッチは、図 8 のように約 20 秒となった。「Thread.sleep();」の引数は「20000」であるため、「20000 ミリ秒」を秒に換算すると「20 秒」となる。つまり、ストップウォッチで測った時間と合致しているため、計測で使うコンピュータの時間の進み方は正しいと判断できる。また、コンピュータの計測結果でも「20000ms」となっていたことから、「Thread.sleep();」や「System.currentTimeMillis()」への解釈違いも起こっていないと判断することができた。

図 7 ストップウォッチを用いた計測方法の正確さ確認のプログラム 図 8 ストップウォッチを用いた計測方法の正確さ確認の結果

```
50 private static void Checkaccuracy(){
51     for(int i=0;i<20;i++){
52         System.out.println(i+"秒前");
53         try {
54             Thread.sleep(1000);
55         }
56         catch (InterruptedException e) {
57         }
58     }
59     System.out.println("開始");
60     long starttime = System.currentTimeMillis();
61     try {
62         Thread.sleep(20000);
63     }
64     catch (InterruptedException e) {
65     }
66     long endtime = System.currentTimeMillis();
67     System.out.println("処理時間:"+ (endtime - starttime)+"ms");
68     System.out.println("終了");
69 }
70
71
```

36 出力 デバッグコンソール ターミナル

```
3秒前
2秒前
1秒前
開始
処理時間:20000ms
終了
```



4. 3 で、計測方法は正確であると判断できたため、本題の「 3×3 メディアンフィルタを使った平滑化」を計測する。

方法 テストプログラムから処理部分を「 3×3 メディアンフィルタを使った平滑化」のメソッドである「noisedelete()」に変更する。「noisedelete()」の引数は、図 9 のように「inputimg」には定義していた「d850429avhrr4.bmp」を入れて、「range」には、 3×3 フィルターを使うため「1」を入れる。

図 9 「 3×3 メディアンフィルタを使った平滑化」の計測プログラム

```
private static void Timemeasure(){
    String fileName = "d850429avhrr4.bmp";
    GImage inputimg= new GImage(fileName);
    int range = 1;

    for(int i=0;i<20;i++){
        long starttime = System.currentTimeMillis();
        noisedelete(inputimg, range);
        long endtime = System.currentTimeMillis();
        System.out.println("開始時刻"+starttime);
        System.out.println("終了時刻"+endtime);
        System.out.println(endtime - starttime);
    }
}
```

2.2 課題 2

2.2.1 計測対象

- 課題 1 と同様に、「 3×3 メディアンフィルタを使った平滑化」を使う。変更するアルゴリズムは、課題 1 で説明した通りにソート部分を変更する。図 10 は、課題 1 で使ったプログラムのソートの部分を抜粋したものである。

図 10 課題 1 プログラムのソート部分

2.2.2 計測方法