算法设计与分析

比较斐波那契不同方法实验报告

一、斐波那契不同解法

1、简单递归方法

利用 f(n)=f(n-1)+f(n-2)的性质直接递归求解，复杂度为 $O(2^n)$

```
public BigInteger recursiveFibonacci(int num) {
        if (num == 0) return BigInteger.ZERO;
        if (num == 1) return BigInteger.ONE;
        return recursiveFibonacci(num-1).add(recursiveFibonacci(num-2));
}
```

2、线性加法

使用线性加法暂存 f(n-1)和 f(n-2)，复杂度为 O(n)

```
public BigInteger bottomUpFibonacci(int num) {
        if (num == 0) return BigInteger.ZERO;
        if (num == 1) return BigInteger.ONE;
        BigInteger n1 = BigInteger.ZERO;
        BigInteger n2 = BigInteger.ONE;
        BigInteger res = BigInteger.ZERO;
        for (int i = 0; i < num-1; i++) {
            res = n1.add(n2);
            n1 = n2;
            n2 = res;
        }
        return res;
}
```

3、矩阵乘法

利用$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$，可以用分治法计算斐波那契数列

```
public BigInteger matrixFibonacci(int num) {
        if (num == 0) return BigInteger.ZERO;
        if (num == 1) return BigInteger.ONE;
        return addMatrix(num-1)[0][0];
}

private BigInteger[][] addMatrix(int num) {
        BigInteger[][] origin= {{BigInteger.ONE, BigInteger.ONE},
                                {BigInteger.ONE, BigInteger.ZERO}};
        if (num == 1) return origin;
        BigInteger[][] a = addMatrix(num/2);
        if (num%2==0)
                return matrixPower(a,a);
        else
                return matrixPower(matrixPower(a,a),origin);
}
```

4、公式法

直接利用斐波那契数列的求值公式计算斐波那契数列的值。因为浮点数运算并不准确，所以在 N 大于一定值后结果会发生错误。

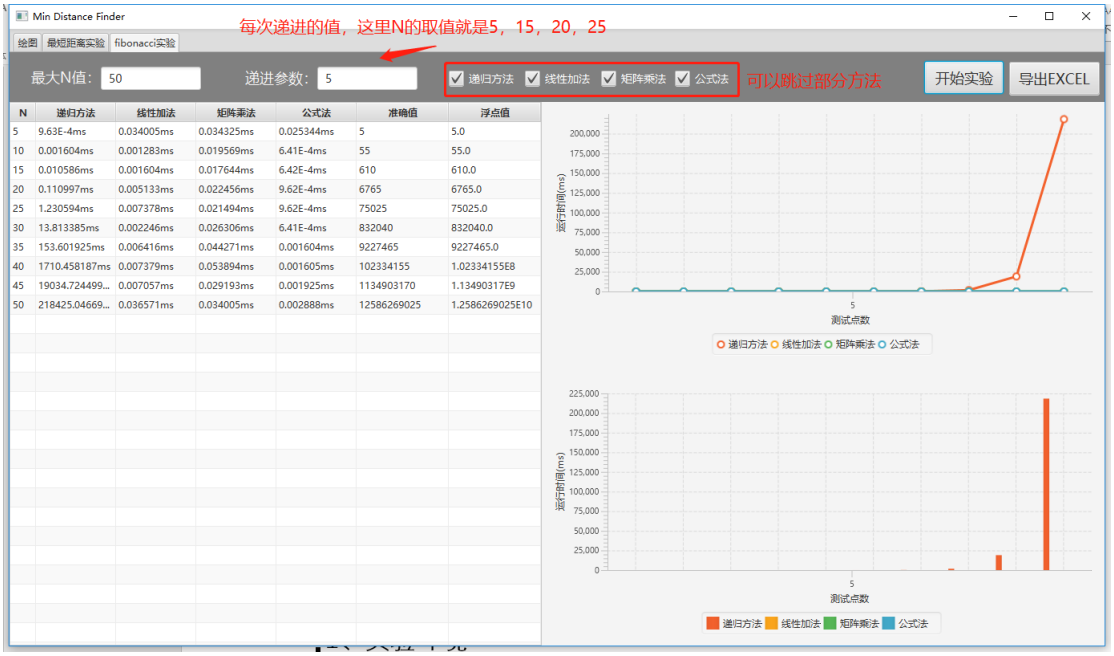
```
public long formulaFibonacci(int num) {
        double n1 = (1 + Math.sqrt(5)) / 2;
        double n2 = (1 - Math.sqrt(5)) / 2;
        double result = ( Math.pow(n1, num) - Math.pow(n2, num) ) / Math.sqrt(5);
        return Math.round(result);
}
```

二、实验设计

1、实验环境

Java 1.8
Windows 10 64 位，　CPU i7-8700

2、实验界面



实验时可以选取实验的最大 N 值和每次实验的递进参数，由于递归方法时间过长，在有些实验中可以跳过。

三、实验结果

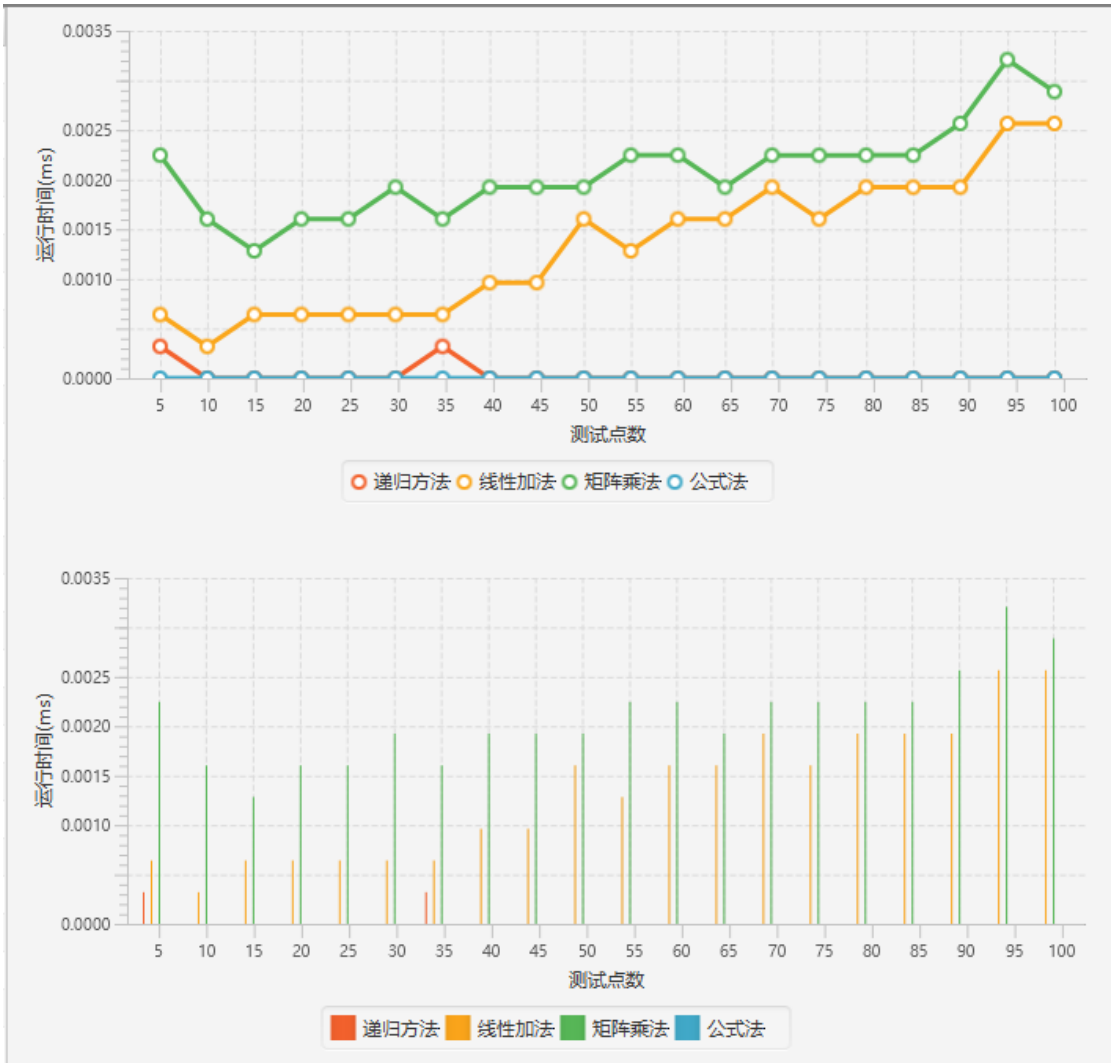
1、运行时间的比较

N	递归方法	线性加法	矩阵乘法	公式法	准确值	浮点值
5	9.63E-4ms	0.034005ms	0.034325ms	0.025344ms	5	5.0
10	0.001604ms	0.001283ms	0.019569ms	6.41E-4ms	55	55.0
15	0.010586ms	0.001604ms	0.017644ms	6.42E-4ms	610	610.0
20	0.110997ms	0.005133ms	0.022456ms	9.62E-4ms	6765	6765.0
25	1.230594ms	0.007378ms	0.021494ms	9.62E-4ms	75025	75025.0
30	13.813385ms	0.002246ms	0.026306ms	6.41E-4ms	832040	832040.0
35	153.601925ms	0.006416ms	0.044271ms	0.001604ms	9227465	9227465.0
40	1710.458187ms	0.007379ms	0.053894ms	0.001605ms	102334155	1.02334155E8
45	19034.724499ms	0.007057ms	0.029193ms	0.001925ms	1134903170	1.13490317E9
50	218425.046692ms	0.036571ms	0.034005ms	0.002888ms	12586269025	1.2586269025E10
60		0.009624ms	0.020531ms	6.42E-4ms	1548008755920	1.54800875592E12
70		0.011869ms	0.020211ms	6.41E-4ms	190392490709135	1.90392490709135E14
80		0.012832ms	0.022777ms	6.41E-4ms	23416728348467685	2.3416728348467744E16

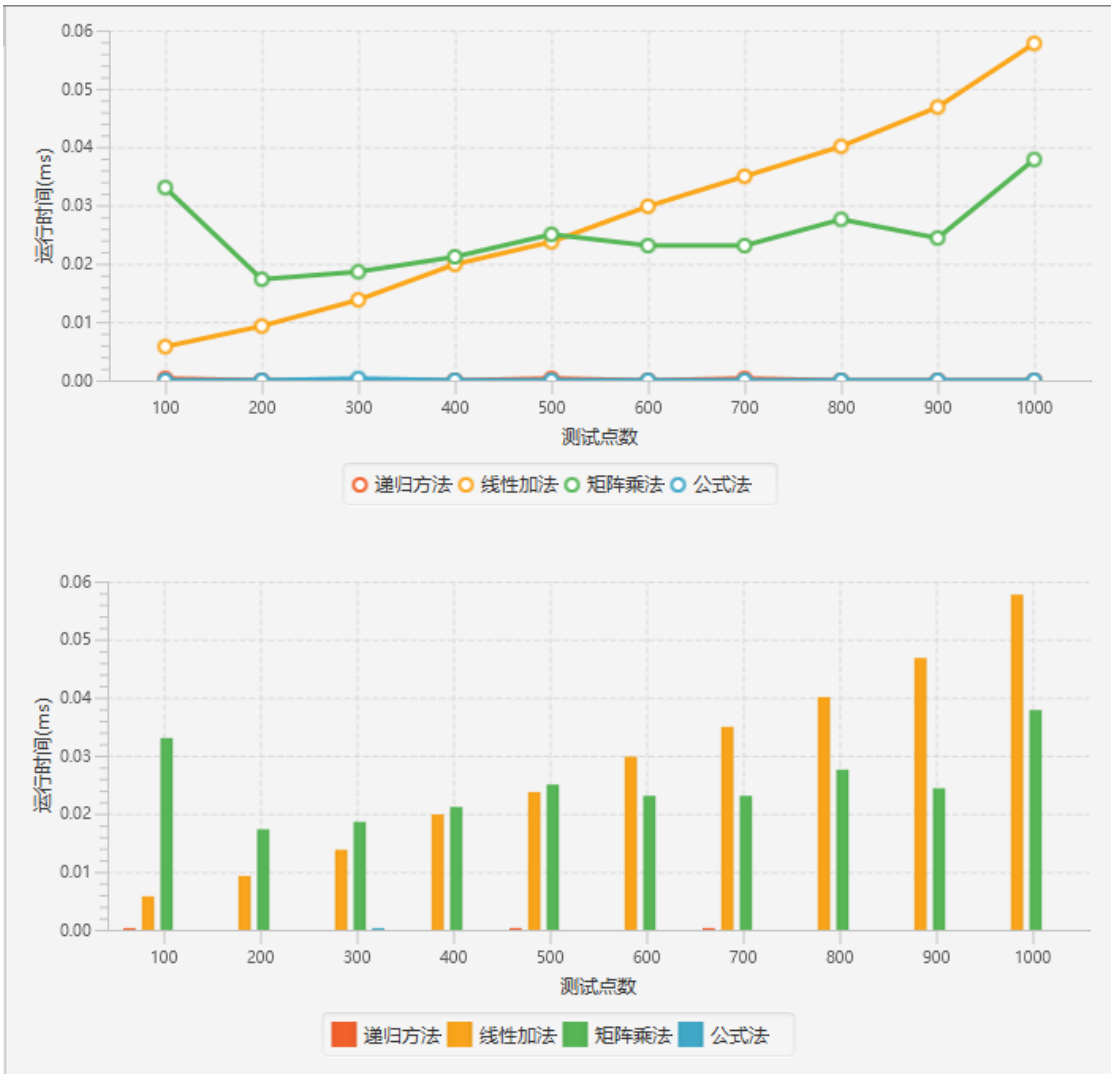
90		0.013153ms	0.020852ms	6.42E-4ms	2880067194370816120	2.8800671943708247E18
100		0.009945ms	0.039138ms	6.41E-4ms	354224848179261915075	
200		0.017965ms	0.041704ms	9.63E-4ms	2805……77189525	
300		0.019889ms	0.039459ms	9.62E-4ms	22223……9600	
400		0.027268ms	0.048762ms	6.41E-4ms	176023……044216019675	
500		0.03625ms	0.05197ms	6.42E-4ms	139423……4125	
600		0.047479ms	0.057744ms	9.63E-4ms	110433070……7901959200	
700		0.065765ms	0.063839ms	9.63E-4ms	8747081……4275	
800		0.049724ms	0.054857ms	6.41E-4ms	69283……0398725	
900		0.066726ms	0.030156ms	9.62E-4ms	5487……800	
1000		0.068651ms	0.027589ms	9.63E-4ms	43466……28875	

因为递归法时间上增长过快，所以不重点关注。虽然公式法的时间复杂度为 O（1），但公式法在 N=80 开始就出现了计算错误。下面重点关注线性加法 O(n)和矩阵乘法 O(lgn)的时间比较。

N 在 5~100 范围内线性加法与矩阵乘法运行时间如图所示(递归和公式法被直接跳过)。

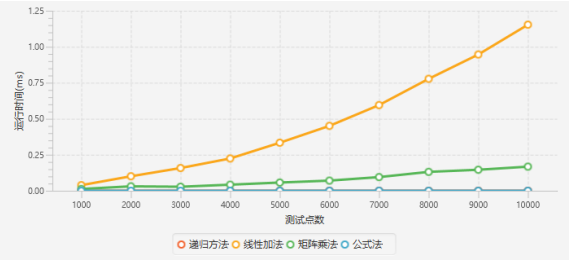


N 在 100 到 1000 的范围内矩阵乘法与线性加法的运行时间如下图所示：



N 取值 10000 时，线性加法的计算时间首次到达 1ms

N	递归方法	线性加法	矩阵乘法	公式法	准确值	浮点值
1000	3.21E-4ms	0.038817ms	0.012511ms	0.0ms	4346655768693...	0.0
2000	0.0ms	0.100732ms	0.030797ms	0.0ms	4224696333392...	0.0
3000	0.0ms	0.157835ms	0.027909ms	0.0ms	4106158863079...	0.0
4000	0.0ms	0.22424ms	0.042346ms	0.0ms	3990947343500...	0.0
5000	0.0ms	0.333633ms	0.057103ms	0.0ms	3878968454388...	0.0
6000	3.21E-4ms	0.450726ms	0.070577ms	0.0ms	3770131493877...	0.0
7000	0.0ms	0.595087ms	0.094957ms	3.21E-4ms	3664348305037...	0.0
8000	0.0ms	0.777301ms	0.131208ms	0.0ms	3561533204460...	0.0
9000	3.21E-4ms	0.946364ms	0.145965ms	0.0ms	3461602912866...	0.0
10000	0.0ms	1.153281ms	0.167458ms	0.0ms	3364476487643...	0.0



2、实验总结

递归方法有极高的算法复杂度 $O(2^n)$，在 N 取值为 50 时计算时间就已经为 218s，难以继续进行实验。

公式法有很稳定的运行时间和极地的算法复杂度 O(1)，但在 N 取值为 80 时就由于浮点数的精度问题出现了计算错误。

线性加法实现简单，算法复杂度为 O(n)，在 N 取值 400 以内时计算时间一直低于矩阵乘法，这是因为他的实现简单，计算中只有加法而没有乘法，提高了运行速度。在 N 取值为

10000 时，线性加法的计算时间到达 1ms。

矩阵乘法理论上算法复杂度为 O(lgn)，运行时间应该低于线性加法，但 N 取值低于 400 时运行时间却一直高于线性加法，在 400 到 500 之间时矩阵乘法与线性加法运行时间相似，直到 N 取值大于 500 时线性加法才开始确定优势，在 N 取值超过 1000 时运行时间开始远远低于线性加法，并且这种差距开始变大。矩阵乘法理论上虽然有更低的算法复杂度，但函数实现复杂，引入了乘法和函数调用，因此在 N 取值较小时难以和线性加法看出差距。