Homework2 实验报告

刘喆骐 2020013163 探微化01

1 实验环境

Windows平台，编程语言为 C++，编译器为 Vscode，使用C++11标准。使用python3进行了绘图，需要安装matplotlib。

2 算法分析

2.1 公式法

斐波那契数列的通项公式为$F_n = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n)$，递推式为$F_n = F_{n-1} + F_{n-2}, F_1 = F_2 = 1$。当n为正整数时，利用快速幂的方法，只需要不断的查看n对应的二进制数的最低位，最低位为1则作为乘积项相乘，否则continue即可。这种方法时间复杂度为$O(logn)$，效率高，但是精度不高，结果不可靠。这是由于参与运算的是浮点数，误差会逐渐积累，且无法取模来减小规模，最终溢出。

2.2 递归法

直接利用递归式$F_n = F_{n-1} + F_{n-2}$，不断递归地计算$F_n = F_{n-1} + F_{n-2}$，其时间复杂度满足$f_n = f_{n-1} + f_{n-2} + 1$，利用归纳法不难证明$2^{\frac{n}{2}} < f_n < 2^n$。不妨假设对于0到n-1此式均成立，那么，$f_n < 2^{n-1} + 2^{n-2} + 1 < 2^n; f_n > 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} + 1 = 2^{\frac{n}{2}}\frac{\sqrt{2}+1}{2} > 2^{\frac{n}{2}}$，得证。此法虽然简单，但是时间复杂度大，非常缓慢，并且消耗大量的栈空间。

2.3 递推法

此法针对递归法进行改进，储存了之前计算的$F(n-1), F(n-2)$，故时间复杂度为$O(n)$，空间复杂度为$O(1)$.

2.4 矩阵法

斐波那契数列递推求解的形式可以看作是进行了矩阵运算，即

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_n & F_{n-1} \\ F_n & F_{n-2} \end{bmatrix}$$

进而有

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

使用快速幂算法即可在$O(lgn)$时间内得到结果。

3. 实验思路

分别使用上述算法编写求斐波那契数列第n项值的程序，通过依次输入不同的n，观察其结果，绘制图线比较得出几种计算方式计算同一n值下斐波那契数列项的用时，说明不同方式的时间复杂度。并比较公式法与其他三种方法的结果，体现公式法计算的误差。

4. 结果分析

4.1 直接递归

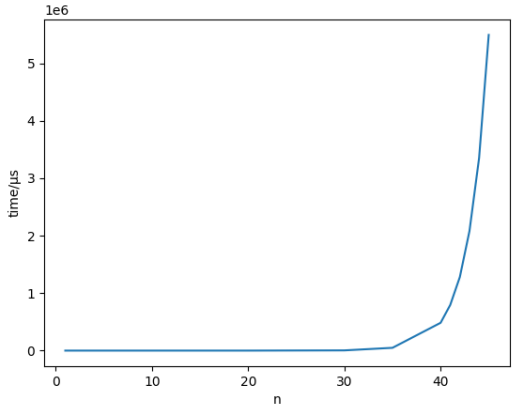

图1 直接递归的时间复杂度

直接递归的时间复杂度远远大于其他组别，出现了显著的指数增长。当n达到43时，运行时间已经超过了2秒，这说明此算法是非常低效的。

4.2 公式法误差分析

误差表如下所示：

n	公式法值	真值
76	598991528	598991529
77	662189183	662189184
78	261180706	261180706
79	923369887	923369890
80	184550580	184550589
81	107920462	107920472
82	292471054	292471061
83	400391524	400391533
84	692862546	692862594
85	93254063	93254120
90	210345270	210345902

表 1：误差对比表（计算过程中均已模 1e9 + 7）

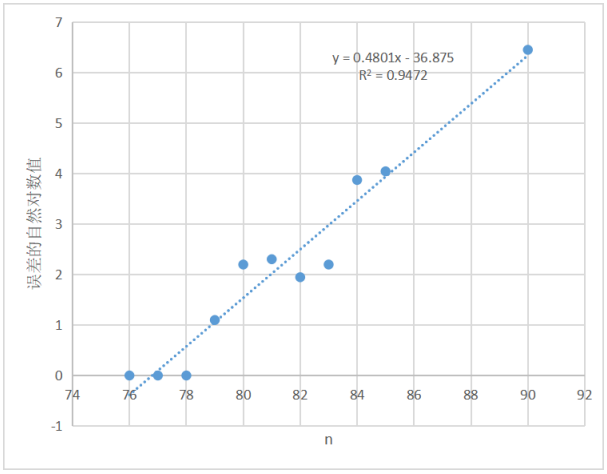

图2 误差的自然对数和n值的关系

可以看到，误差和n近似成指数关系，

$$error = e^{0.4801x-36.875}, R^2 = 0.9472,$$

相关性较好。

4.3 时间复杂度比较

时间消耗表如下：

n	递归	公式	递推	矩阵
1	0.1	0.3	0.1	0.1
10	0.6	0.3	0.1	0.1
50	5.95E+07	0.5	0.4	0.5
100	nan	0.5	0.7	0.5
1000	nan	0.4	6.3	0.6
10000	nan	0.4	67.3	0.4
100000	nan	0.8	868.4	0.9
1000000	nan	0.5	8098.2	0.8
10000000	nan	0.7	69450.6	0.6
100000000	nan	0.5	671761	1
100000000	nan	0.5	671761	1
200000000	nan	1.2	1.35E+06	1.2
300000000	nan	0.5	2.01E+06	1.1
400000000	nan	0.5	2.71E+06	0.8
500000000	nan	0.6	3.37E+06	0.8
600000000	nan	0.7	4.06E+06	1.2
700000000	nan	2	4.74E+06	0.9
800000000	nan	0.6	5.41E+06	0.7
900000000	nan	0.7	6.09E+06	1.5
1000000000	nan	0.6	6.73E+06	1.2

表2 各方法耗时(微秒)和n的关系

可以看到，递归耗时显著长，而递推大致成线性时间，公式法和矩阵的耗时小暂时看不出规律。对于递推法进行线性回归，如图：

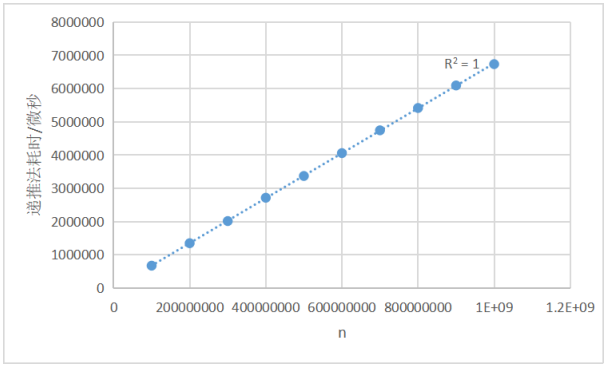

图3 递归法时间线性回归

拟合公式为

$$y = 0.0068x - 2220.3$$

$R^2 = 1$,发现有很好的线性性，证明了递推法时间复杂度为线性.

4.4 矩阵法时间复杂度

为了看出矩阵法的时间复杂度，采用更大是数进行实验。并绘制logn-耗时图，如下：

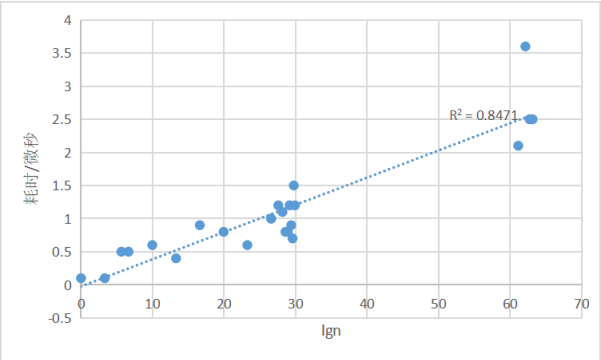

图4 logn-耗时图

可以看出，该图并非完美的线性，依旧有很大的波动，这可能是由于不同的n的二进制数中的1的数量不同，1越多，所需的运算越多，时间越长。

5. 小结

本次实验通过编程，数据采集和绘图分析，直观地体现了公式法计算斐波那契数列的误差、并拟合得出误差增长的近似函数。通过算法分析推导出公式法、递归法、递推法、矩阵法的时间复杂度，并绘图、进行曲线拟合验证了此前的推导。

利用矩阵和快速幂的结合，可以在保证计算精度的情况下达到$O(lgn)$的时间复杂度，是此问题的最好算法。