计算斐波拉契数实验报告

姓名：覃果　学号： 2020012379　班级：软件02

2022 年 3 月 2 日

摘要

通过暴力递归，分治计算数字的幂，自底向上以及分治计算矩阵的幂四种方法来计算斐波拉契数，并分析每种方法的时间，空间复杂度以及通过测试比较不同算法的实际运行时间，从而加深对算法复杂度的认识。

1　实验环境

CPU: AMD Ryzen 5 4600U with Radeon Graphics 2.10 GHz
OS: Windows 10 家庭中文版 20H2
Python interpreter: Python 3.8.7

2　算法分析

2.1　暴力递归

2.1.1　算法描述

FIBONNACI_1(n):
if n = 0: return 0
if n = 1: return 1
return FIBONNACI_1(n - 1) + FIBONNACI_1(n - 2)

2.1.2　算法分析

时间复杂度：
由算法过程知：$T(n) = T(n-1) + T(n-2) \Rightarrow T(n) = \Omega(\phi^n), \phi = \frac{\sqrt{5}+1}{2}$
空间复杂度：
为函数栈的最大深度 $O(n)$

算法评价：

递归过程中进行了大量的重复计算，时间复杂度大大增加。指数级别复杂度不可接受。

2.2 分治计算数字的幂

2.2.1 算法描述

$\phi = \frac{\sqrt{5}+1}{2}$

FIBONNACI_2(n):

return $\lfloor \frac{power(\phi,n)}{\sqrt{5}} + 0.5 \rfloor$

其中$power(\phi, n)$以分治的方法，先计算$power(\phi, \frac{n}{2})$，再平方

2.2.2 算法分析

时间复杂度：

由算法过程知：$T(n) = T(\frac{n}{2}) + \Theta(1) \Rightarrow T(n) = \Theta(lgn)$

空间复杂度：

为函数栈的最大深度 $O(lgn)$

算法评价：

利用了分治的思想，大大简化了运算，算法的时间复杂度以及空间复杂度都很理想，但是由于浮点数运算误差的逐渐累积，当n较大时，会产生较大的绝对误差。

2.3 自底向上

2.3.1 算法描述

FIBONNACI_3(n):

if n = 0:

return 0

if n = 1:

return 1

for i : 1 → n

if i = 0: array[0] = 0

if i = 1: array[1] = 1

array[i] = array[i - 1] + array[i - 2]

return array[n]

2.3.2 算法分析

时间复杂度：

由算法过程知：$T(n) = \Theta(n)$

空间复杂度：

为数组的长度 $O(n)$

算法评价：利用了自底向上的设计思想，算法简洁明了，易于理解和实现。时间复杂度和空间复杂度都可以接受，但仍有优化空间。

2.4 分治计算矩阵的幂

2.4.1 算法描述

$$matrix = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

FIBONNACI_4(n):

return power(matrix, n + 1)[1][1]

其中power(matrix, n + 1),以分治的方法，先计算power(matrix, $\frac{n+1}{2}$))，再平方

2.4.2 算法分析

时间复杂度：

由算法过程知：$T(n) = T(\frac{n}{2}) + \Theta(1) \Rightarrow T(n) = \Theta(lgn)$

空间复杂度：

为函数栈的深度 $O(lgn)$

算法评价：巧妙利用了数学定理以及分治算法，算法时间空间复杂度极优，且不会有浮点数的计算误差。

3 实验设计思路

因为本次算法比较简单，直接根据上述算法描述即可完成对应的代码编写

同时，对四个算法分别测试不同样例下的运行时间，并分析结果如下

4 结果分析

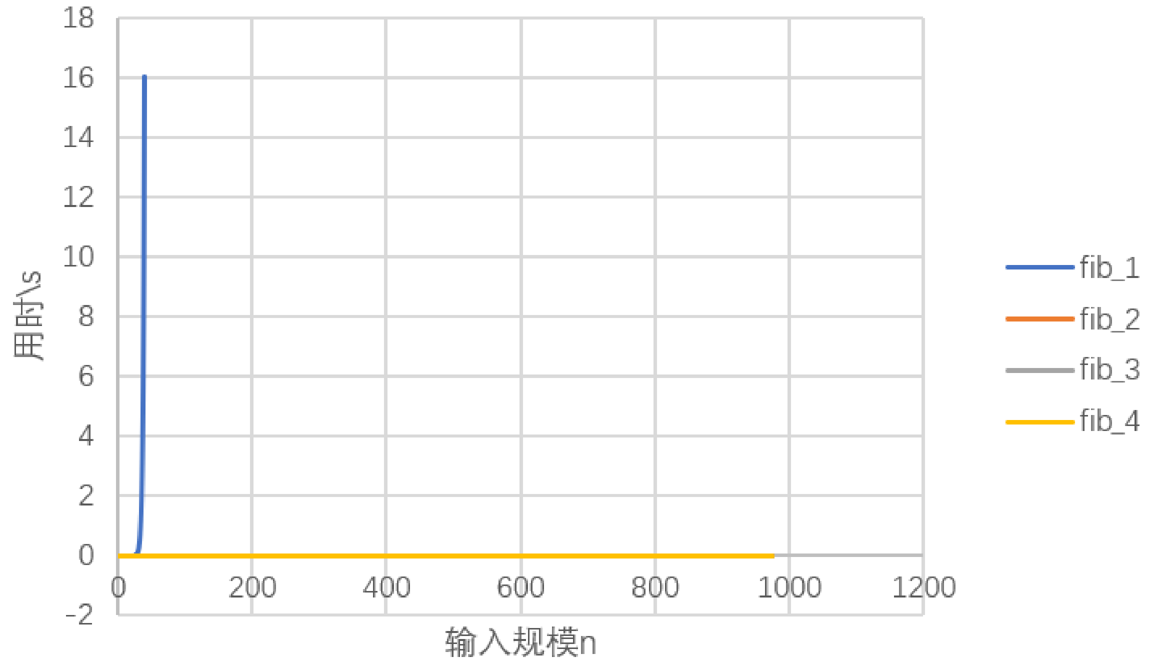

图 1: 结果分析

可见，算法1用时显著高于其他算法，而其他三种算法在该输入规模下用时基本一致

同时，分析算法二的绝对误差随输入规模的变化，如下图

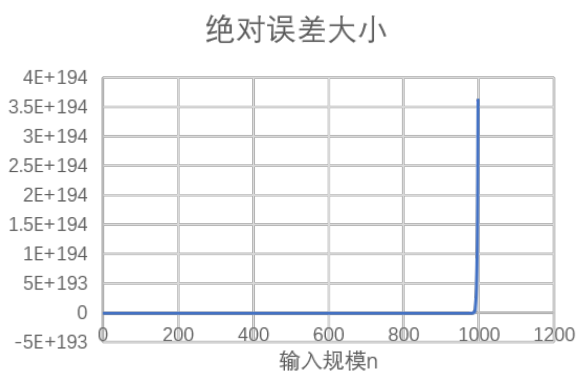

图 2: 绝对误差