斐波那契数列计算方法比较实验报告

软件13 杨楠 2021010711

摘要：本报告对比分析了求斐波那契数列的第n项的几种算法，着重分析算法复杂度，并通过编写代码，测试比较运行时间。递归法的时间复杂度最大，为指数级。递推法的时间复杂度为线性。代入公式计算法和矩阵乘法的时间复杂度均为对数级，但是前者的计算会存在误差。

# 1 实验环境

使用C++11语言在VSCode平台编写代码，在Windows环境编译运行。

# 2 算法分析

斐波那契数列的递归式定义如下：

由递归式，可以用特征方程法或者数学归纳法，解得的通项公式为

其中

更具体地，由于，从而

所以[1]

## 2.1 递归法

直接利用递归式(1)编写递归函数：欲求F(n)，需调用自身函数F(n-1)和F(n-2)，直至递归终点（即n==1和n==0）。

递归法的T(n)满足(1)式，即

相应地可以求出

为指数复杂度。

## 2.2 递推法

由递归式(1)，也可直接用递推法进行循环迭代，依次求出F(0), F(1), …, F(n)

每次循环内进行有限次操作，从而

为线性复杂度。

## 2.3 公式法

由(2)式或者(3)式，可以直接将n代入计算，得出结果。

其中的乘方计算，如果用逐次相乘，乘n次，那么时间复杂度为。

而如果使用快速幂的方法，即：欲求，先求出，以此递归下去，那么有

使用主定理，可算得

## 2.4 矩阵乘法

可以用数学归纳法证明

证明如下。当时，上式成立，即

假设时，上式成立，即

那么当时，有

又有

从而成立。综上，(4)式成立。

由此可以使用矩阵乘法，算出从而得到的值。矩阵乘法如果使用逐次相乘计算，时间复杂度为，如果用快速幂，类似于之前的公式法，可算得

# 3 实验设计思路

计算斐波那契具体项的函数，都命名为F(n)。

long long F(long long n)

使用long long类型，是便于后续数据大的情况下的调用。

测试数据过大时，计算结果可能溢出。可以通过取模的方式规避，但是本实验主要是比较算法所用时间，所以对此不作过多的处理。

分别在demo1.cpp，demo2.cpp，demo3-1.cpp，demo3-2.cpp，demo4.cpp中测试这几种算法。由于不同算法的时间复杂度差距较大，所以不直接同时运行，而是分开测试。

具体测试时，直接调用函数test()，可以根据需要打印出运行时间（单位是微秒）等其他数据。

void test(long long n)

{

    cout << n;

    cout << " ";

    auto start = steady\_clock::now();

    long long result = F(n);

    auto end = steady\_clock::now();

    duration<double, micro> diff = end - start;

    cout << diff.count();

    cout << " ";

    cout << result;

    cout << " ";

    cout << "\n";

}

使用了<chrono>库，用于计时。调用F()之前获取当前时间start=steady\_clock::now()，调用完毕后再次获取当前时间end，两者作差得到diff。

## 3.1 递归法

函数如下。

long long F(long long n)

{

    if(n == 0)

        return 0;

    else if(n == 1)

        return 1;

    else

        return F(n-1) + F(n-2);

}

## 3.2 递推法

函数如下。

long long F(long long n)

{

    long long a = 0, b = 1;

    if(n == 0)

        return a;

    else

    {

        for(long long i = 2; i <= n; i++)

        {

            b = a + b;

            a = b - a;

        }

        return b;

    }

}

每次循环，更新a和b，最终返回b的值。

## 3.3 公式法

函数如下。

long long F(long long n)

{

    double phi = (sqrt5 + 1) / 2;

    long long result = round(power(phi, n) / sqrt5);

    return result;

}

其中，调用了<cmath>库中的round()函数，用于获取距离该数最近的整数（即四舍五入）。在power()函数中使用快速幂的方式。常量sqrt5的值为sqrt(5)。

## 3.4 矩阵乘法

函数如下。

long long F(long long n)

{

    matrix m{1, 1, 1, 0};

    matrix result = power(m, n);

    return result.a12;

}

其中matrix是一个结构体，效果是一个2\*2矩阵。用power()函数实现矩阵乘方运算。

# 4 结果分析

## 4.1 递归法

编译demo1.cpp后运行，将输出结果整理成表后，绘图如下。

可以看出，当n比较大时，lgT与n大致是线性关系，从而可知T确实是呈指数增长。

注意到当n>=40后，T有了很明显的上升，在n>=45后，T基本是在1秒甚至10秒以上。可以预见，当n更大的时候，T将会增长地更快。

## 4.2 递推法

编译demo2.cpp后运行，将输出结果整理成表后，绘图如下。

可以看到T与n大致呈线性关系，从而验证成立。

## 4.3 公式法

### 4.3.1 运行时间

编译demo3-1.cpp后运行，将输出结果整理成表后，绘图如下。

可以看到当n较大时，T与lgn大致呈线性关系，从而验证成立。

### 4.3.2 计算误差

由于C++对于无理数是用浮点数double进行存储的，在计算过程中又进行了乘法运算，计算结果可能会出现误差。

编译demo3-2.cpp后运行，将输出结果整理成表格如下。

|  |  |  |  |
| --- | --- | --- | --- |
| n | F(n) | F'(n) | F'(n)-F(n) |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |
| 2 | 1 | 1 | 0 |
| 3 | 2 | 2 | 0 |
| 4 | 3 | 3 | 0 |
| … | … | … | … |
| 70 | 190392490709135 | 190392490709135 | 0 |
| 71 | 308061521170129 | 308061521170129 | 0 |
| 72 | 498454011879264 | 498454011879264 | 0 |
| 73 | 806515533049393 | 806515533049393 | 0 |
| 74 | 1304969544928657 | 1304969544928657 | 0 |
| 75 | 2111485077978050 | 2111485077978050 | 0 |
| 76 | 3416454622906707 | 3416454622906708 | 1 |
| 77 | 5527939700884757 | 5527939700884758 | 1 |
| 78 | 8944394323791464 | 8944394323791466 | 2 |
| 79 | 14472334024676221 | 14472334024676226 | 5 |
| 80 | 23416728348467685 | 23416728348467700 | 15 |
| 81 | 37889062373143906 | 37889062373143928 | 22 |
| 82 | 61305790721611591 | 61305790721611624 | 33 |
| 83 | 99194853094755497 | 99194853094755552 | 55 |
| 84 | 160500643816367088 | 160500643816367232 | 144 |
| 85 | 259695496911122585 | 259695496911122816 | 231 |
| 86 | 420196140727489673 | 420196140727490112 | 439 |
| 87 | 679891637638612258 | 679891637638612992 | 734 |
| 88 | 1100087778366101931 | 1100087778366103296 | 1365 |
| 89 | 1779979416004714189 | 1779979416004716288 | 2099 |
| 90 | 2880067194370816120 | 2880067194370819584 | 3464 |
| 91 | 4660046610375530309 | 4660046610375535616 | 5307 |
| 92 | 7540113804746346429 | 7540113804746356736 | 10307 |

用F(n)表示理论值，F’(n)表示计算值。可以看到，在n<=75的时候，两者相等，但n>=76时，计算值与理论值开始有偏差，计算值偏大些。并且随n增大而增大。说明用这种方法计算，确实会存在计算误差。

## 4.4 矩阵乘法

编译demo4.cpp后运行，将输出结果整理成表后，绘图如下。

可以看到当n较大时，T与lgn基本呈线性关系，从而验证成立。

# 5 结论

比较以上四种算法可得：递归法尽管思路非常自然，但是时间复杂度是指数级，运行效率非常有限，在n>=40之后的计算所耗费的时间就很高。递推法尽管是递归法的一个逆运算，但是时间复杂度降到了线性。实际上，如果将递归函数进行改进，比如使用数组来保存已经求得的前几项的值，下一次调用的时候直接使用而，则也可将时间复杂度降为线性。

公式法和矩阵乘法的时间复杂度都是对数级，但是由于前者设计根号的乘方运算，会产生误差，其准确性不如后者。综合来看，使用矩阵乘法快速幂，在准确性和效率上，是这几种方法中较高的。

本实验着重比较不同算法的时间复杂度，当数据较大时（比如n>=100000），计算结果会溢出。尽管可以使用大数加法（高精度加法）等方式进行优化，但这些方式本身也会带来时间复杂度，本实验在此方面不做过多考察。

# 参考文献

1. CLRS, Introduction to Algorithms (3rd edition), (2009), The MIT Press