作业 1 Fibonacci 数列问题

解雲暄 3190105871

1 问题描述

定义 Fibonacci 数列：$Fib(n) = Fib(n-1) + Fib(n-2), Fib(0) = 0, Fib(1) = 1$。

分别直接利用递推公式和递归法来实现计算 Fibonacci 数列第 n 项，分析时间复杂度并用程序耗时验证。

2 算法描述与代码设计

该问题算法较为简单，代码中包含详细注释，在此不再赘述。

代码中有 `typedef unsigned long long ull;`。

2.1 递推法计算：

```
ull fibIter(int n) {
    // 防止出现 n 小于 0 情况
    if (n < 0) {
        cout << "Error! n < 0" << endl;
        return 0;
    }
    // f(0) = 0, f(1) = 1
    if (n <= 1)     return n;

    // 递推关系：fib_2 表示当前正在计算的项目，fib_1 和 fib_0 分别是前面两项
    // 在每次计算时，上一次的 fib_2 和 fib_1 成为下一次的 fib_1 和 fib_0
    // fib_0 = f(0) = 0, fib_1 = f(1) = 1
    ull fib_0 = 0, fib_1 = 1, fib_2 = 1;
```

```
    for (int i = 3; i <= n; i++) {
        fib_0 = fib_1;
        fib_1 = fib_2;
        fib_2 = fib_0 + fib_1;
    }
    return fib_2;
}
```

尝试对遍历过程做简化：

```
    ull fib[3] = { 0, 1, 1 };
    for (int i = 3; i <= n; i++) {
        fib[i % 3] = fib[(i + 1) % 3] + fib[(i + 2) % 3];
    }
    return fib[n % 3];
```

2.2 递归法计算：

```
ull fibRecr(int n) {
    if (n < 0) {
        cout << "Error! n < 0" << endl;
        return 0;
    }
    if (n <= 1)      return n;
    // 递归关系式
    return fibRecr(n - 1) + fibRecr(n - 2);
}
```

上述三种算法分别标号为函数 $1, 2, 3$。

3 算法分析

3.1 分析递推法用时

对于递推法，耗时操作为循环。循环次数是 $O(n)$ 的，循环内有常数次运算和赋值，因此递推法的时间复杂度是 $O(n)$ 的。

3.2 分析递归法用时

记 $T(i)$ 为计算 $Fib(i)$ 所用时间，那么计算 $Fib(i)$ 时，我们需要分别计算 $Fib(i-1)$ 和 $Fib(i-2)$，因此有递推关系：

$$T(i)=T(i-1)+T(i-2)+C\geq T(i-1)+T(i-2)\triangleq t(i)$$

(C 是其他操作消耗的常量时间；为了计算方便，我们忽略该项并用 $t(i)$ 表示用时。)

那么

$$\begin{aligned}t(n)&=t(n-1)+t(n-2)\\&=(t(n-2)+t(n-3))+t(n-2)\\&=2\cdot t(n-2)+t(n-3)\\&=(2\cdot t(n-3)+2\cdot t(n-4))+t(n-3)\\&=3\cdot t(n-3)+2\cdot t(n-4)\\&=\dots\\&=Fib(i)\cdot t(n-i)+Fib(i-1)\cdot t(n-i-1)\\&=\dots\\&=Fib(n-1)\cdot t(1)+Fib(n-2)\cdot t(0)\\&=Fib(n-1)+Fib(n-2)\\&=Fib(n)\end{aligned}$$

即有 $T(n)\geq t(n)=O(Fib(n))$。

再推导 Fibonacci 数列的通项公式。根据 $Fib(n)=Fib(n-1)+Fib(n-2)$，设

$$Fib(n)+a\cdot Fib(n-1)=b\cdot(Fib(n-1)+a\cdot Fib(n-2))$$

则有

$$Fib(n)=(b-a)Fib(n-1)+ab\cdot Fib(n-2)$$

即 $b-a=ab=1$，即有 $\frac{1}{a}-a=1$，即 $a^2+a-1=0$，解得 $a=\frac{-1\pm\sqrt{5}}{2},b=\frac{1\pm\sqrt{5}}{2}$。

因此有

$$\begin{aligned}Fib(n)+a\cdot Fib(n-1)&=b\cdot(Fib(n-1)+a\cdot Fib(n-2))\\&=b^2\cdot(Fib(n-2)+a\cdot Fib(n-3))\\&=b^{n-1}\cdot(Fib(1)+a\cdot Fib(0))=b^{n-1}\end{aligned}$$

即有

$$\begin{cases}Fib(n+1)+\frac{-1+\sqrt{5}}{2}\cdot Fib(n)=(\frac{1+\sqrt{5}}{2})^n\\Fib(n+1)+\frac{-1-\sqrt{5}}{2}\cdot Fib(n)=(\frac{1-\sqrt{5}}{2})^n\end{cases}$$

两式相减得 $\sqrt{5}\cdot Fib(n)=(\frac{-1+\sqrt{5}}{2})^{n}-(\frac{-1-\sqrt{5}}{2})^{n}$，即有

$$Fib(n)=\frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^{n}-(\frac{1-\sqrt{5}}{2})^{n}]$$

综上，时间复杂度为 $O(\frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^{n}-(\frac{1-\sqrt{5}}{2})^{n}])$。

查阅资料知，$(\frac{3}{2})^n \leq Fib(n) \leq (\frac{5}{2})^n$，即该算法的时间复杂度是指数级的。

4 测试与结果分析

4.1 测试设计与结果

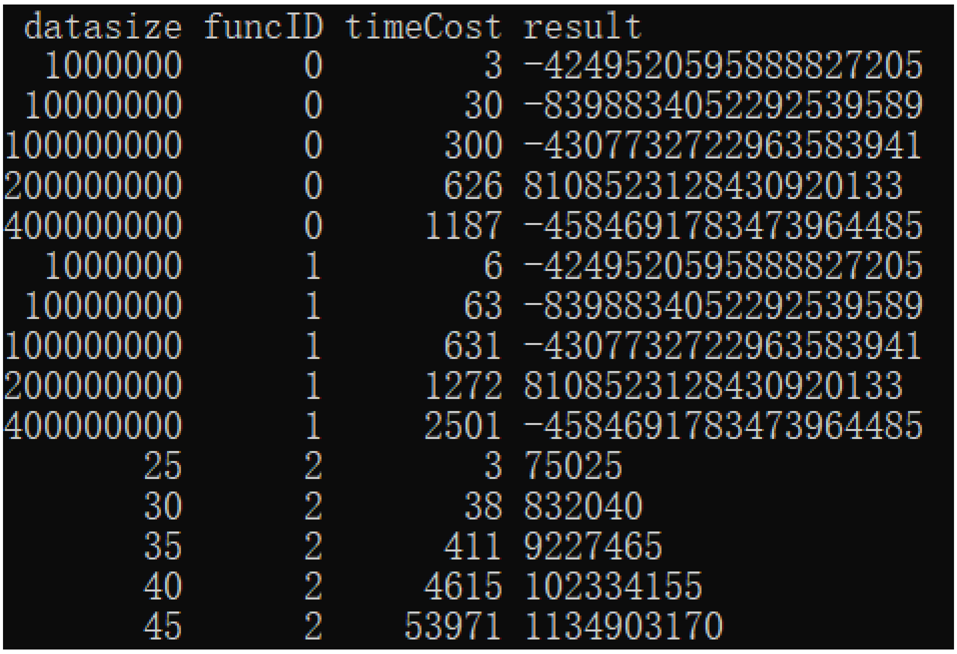
根据上述分析以及一些前期测试，我们设计了两组数据规模用于测试。对于某次测试，数据规模为 x 表示我们需要求 $Fib(x)$ 的值。

第一组较大的数据 `[1000000, 10000000, 100000000, 200000000, 400000000]` 用于递推法的测试；第二组较小的数据 `[25, 30, 35, 40, 45]` 用于递归法的测试。测试代码如下：

```
int main()
{
    // 计时
    clock_t start, end;

    const int DATASIZE_NUM = 5, FUN_NUM = 3;
    // 测试数据，分别设置了两组各 5 个数据作为所求项数
    const int datasize[2][DATASIZE_NUM] =
    { { 1000000, 10000000, 100000000, 200000000, 400000000 },
      { 25, 30, 35, 40, 45 } };
    // 测试函数，将三个待测函数放到函数指针数组中
    ull(*fun[FUN_NUM])(int) = { fibIter , fibIterOptm, fibRecr };

    cout << " datasize funcID timeCost result" << endl;

    for (int j = 0; j < FUN_NUM; j++)    // 分别测试 3 种函数
        for (int i = 0; i < DATASIZE_NUM; i++)
        {
            start = clock();     // 开始计时
            // 对于前两种递推函数，采用第一组较大数据测试；
            // 对于第三种递归函数，采用第二组较小数据测试。
```

```
            ull result = fun[j](datasize[j < 2 ? 0 : 1][i]);
            end = clock();      // 结束计时
            // 输出数据规模、函数编号、用时和结果
            printf("%9d %6d %8d %lld\n", datasize[j < 2 ? 0 : 1][i], j, end - start, result);
        }

    return 0;
}
```

测试结果截图如下:



datasize	funcID	timeCost	result
1000000	0	3	-4249520595888827205
10000000	0	30	-8398834052292539589
100000000	0	300	-4307732722963583941
200000000	0	626	8108523128430920133
400000000	0	1187	-4584691783473964485
1000000	1	6	-4249520595888827205
10000000	1	63	-8398834052292539589
100000000	1	631	-4307732722963583941
200000000	1	1272	8108523128430920133
400000000	1	2501	-4584691783473964485
25	2	3	75025
30	2	38	832040
35	2	411	9227465
40	2	4615	102334155
45	2	53971	1134903170

测试结果。函数 1, 2 的数据范围较大，结果均发生了溢出

4.2 递推法结果分析

递推法采用数据较大，因此结果均已溢出。下面 t_1 和 t_2 分别是函数 1, 2 的测试结果:

n	time cost 1 t_1	$10^6 \cdot \frac{t_1}{n}$	time cost 2 t_2	$10^6 \cdot \frac{t_2}{n}$
10^6	3	3.00	6	6.00
10^7	30	3.00	63	6.30
10^8	300	3.00	631	6.31
2×10^8	626	3.13	1272	6.36
4×10^8	1187	2.97	2501	6.25

可以看到，两个函数 $10^6 \cdot \dfrac{t}{n}$ 的值均基本一致，这可以验证我们 $t(n)=O(n)$ 的结论。

同时横向比较，我们发现函数 2 的耗时是同等数据规模下函数 1 的 2 倍。这说明，虽然函数 2 节省了赋值操作，但是增加的数组寻址和取模运算（尤其是后者）带来了更多的时间消耗。

4.3 递归法结果分析

递归采用的是较小的数据。可以看到，递归法用时显著高于递推法。我们在第 3 节中已经分析了原因。

n	**time cost** t	$Fib(n)$	$10^5 \cdot \dfrac{t}{Fib(n)}$	$10^5 \cdot \dfrac{t}{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}$
25	3	75025	4.00	1.79
30	38	832040	4.57	2.04
35	411	9227465	4.45	1.99
40	4615	102334155	4.51	2.02
45	53971	1134903170	4.76	2.13

在该数据范围内，$Fib(n)$ 的值并未超出 `unsigned long long` 的范围，我们可以利用该值进行进一步分析。我们可以看到，$10^5 \cdot \dfrac{t}{Fib(n)}$ 的值基本一致，这验证了我们在第 3 节中理论推导出的 $t(n)=O(Fib(n))$ 的结论。

我们进一步计算 $10^5 \cdot \dfrac{t}{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}$，发现其值也基本一致，且都满足 $10^5 \cdot \dfrac{t}{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n} = \dfrac{1}{\sqrt{5}} 10^5 \cdot \dfrac{t}{Fib(n)}$。这证明了我们第 3 节后续推导及最终结论的正确性。