实验2附加题题解

1. 快速幂

   我们观察代码，每次除以二判断余数，其实是一个对n进行二进制分解的框架。

   另外仔细观察发现，整个while循环的时候，a的值变为1次、2次、4次、8次……

   这个次数正好是2的整次幂。当n的二进制在第k位为1的时候，ans会将a的2的k次方乘进来。

   例如，n=139，转换在二进制的权值分解是128+8+2+1，也就是在7、3、1、0位有1，因此这个程序中ans会变成：

   $$ans = a^{2^0} * a^{2^1} * a^{2^3} * a^{2^7} = a^{1+2+8+128} = a^{139}$$

   至于这么做的好处，如果我们用一个1到n的循环将a累乘n次，那么这个程序的执行时间是和n线性相关的；但如果用上面这种方式，由于一个数的二进制位数是对数级别的，这个程序的执行时间是和logn（根据换底公式，底数只有系数的差别，并不重要）线性相关。或者说，程序的复杂度从**o(n)**降到了**o(logn)**。直观来说，当n增加一倍，前者也要增加一倍的开销，但是后者只用多循环一次。

2. 上楼梯

   以第一个问题为例子，在第k级楼梯的走法可以分成两种：一种是从k-1级跨一步上来，另一种是k-2级跨两步上来。这两类的走法肯定不会有重复，因为他们的最后一步是不同的。并且，它们一定涵盖了到第k级的所有走法。

   因此，得到一条递推的式子：

   $$f_k = f_{k-1} + f_{k-2}$$
   $$f_1 = 1, f_2 = 2$$

   用数组实现即可。

   同样地，变成1，2，4步，我们一样有：

   $$f_k = f_{k-1} + f_{k-2} + f_{k-4}$$
   $$f_1 = 1, f_2 = 2, f_3 = 3, f_4 = 6$$

3. 结合两个问题

$$
\begin{aligned}
f_i &= a_1 f_{i-1} + a_2 fi - 2 + \cdots + a_k f_{i-k} \\
&= \sum_{j=1}^{k} a_i f_{i-j}
\end{aligned}
$$

这样的数列每一项都是前k项的线性相加，我们成为**常系数线性递推数列**。

一个比较著名的例子就是斐波那契数列。

对于这种数列，我们都可以得到这样的一个转移矩阵。以上题第二小题为例子：

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

可以看出，前三列中，A i,i+1 =1，其他都为0。最后一列，A k,i =ai

我们计算一下下面的式子：

$$
\begin{bmatrix} f_{i-4} & f_{i-3} & f_{i-2} & f_{i-1} \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} f_{i-3} & f_{i-2} & f_{i-1} & f_i \end{bmatrix}
$$

乘这个矩阵一次，就能得到下一项。

因此我们计算：

$$
\begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix} \times A^{n-4} = \begin{bmatrix} f_{n-3} & f_{n-2} & f_{n-1} & f_n \end{bmatrix}
$$

因此我们计算出矩阵A的n-4次方就行了。

那么，如果我们能用第一题的方法来计算矩阵的n-4次方，就能在o(logn)的时间复杂度下，得到数列的第n项。

具体的实现方式，大家在学习了如何将数组作为函数参数之后可以实现；自学了运算符重载之后可能有更加整洁的代码实现方式。

但是矩阵的乘法代码有三重循环，复杂度是行列的三次方级别的，有时候可能无法接受。下面给出一些参考资料，运用了多项式、快速傅立叶变换等方法优化这方面。大家有兴趣可以看看。

[Zhang_RQ]常系数齐次线性递推初探

luoguP4732@github

luoguP3824 登陆看题解