

## 内容：

- 算法
- 实现
- 应用
  - 带模求幂
  - 斐波那契数
  - 施加同一置换k次
  - 坐标系中点的操作
  - 图上长度为k的路径数
  - 快速幂变种：大数相乘模m
- 练习题

快速幂（binary exponentiation）是一种  $O(\log n)$  时间复杂度求  $a^n$  的算法

事实上快速幂可以求解很多和代数运算无关的问题中，这类问题的共同点是操作具有 **结合律**：

$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$$

下面就从最基础的求幂问题开始介绍快速幂算法，也会讨论到矩阵乘法等其他问题

## 算法

朴素算法中求  $a^n$  需要进行  $n - 1$  次乘法操作  $a^n = a \cdot a \cdots a$ ，这并不适合  $n$  很大的情况。但求幂过程中可以对指数做如下拆分：

$$a^{b+c} = a^b \cdot a^c, \quad a^{2b} = a^b \cdot a^b = (a^b)^2$$

快速幂的核心是把求幂过程按指数的二进制位进行拆分，比如：

$$3^{13} = 3^{(1101)_2} = 3^8 \cdot 3^4 \cdot 3^1$$

既然任意一个数  $n$  的二进制表示有  $\lfloor \log_2 n \rfloor$  位，如果我们已知每一位上求幂的结果（ $a^1, a^2, a^4, a^8, \dots, a^{2^{\lfloor \log_2 n \rfloor}}$ ），那么我们也只需要  $O(\log n)$  次乘法就能得到  $a^n$ ，所需条件也不难求得：

$$\begin{aligned} 3^1 &= 3 \\ 3^2 &= (3^1)^2 = 3^2 = 9 \\ 3^4 &= (3^2)^2 = 9^2 = 81 \\ 3^8 &= (3^4)^2 = 81^2 = 6561 \end{aligned}$$

为了回答  $3^{13}$  的结果，我们只需要把三个数乘起来就可以了（跳过  $3^2$  因为对应的二进制位为 0）：  
 $3^{13} = 6561 \cdot 81 \cdot 3 = 1594323$

下面的递归式描述了同样的思想：

$$a^n = \begin{cases} 1 & \text{if } n == 0 \\ (a^{n/2}) & \text{if } n > 0 \text{ and } n \text{ even} \\ (a^{n/2}) \cdot a & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases}$$

## 实现

直接翻译递归式我们可以得到一个递归的实现：

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

第二种方式用循环实现，尽管复杂度相同，但是在实际运行中会比递归的方式更快：

```
long long binpow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

## 应用

### 带模求幂

问题：求  $x^n \bmod m$ ，更常见的一个场景是求  $x^{-1} \bmod m$

做法： $a \cdot b$  和  $(a \bmod m) \cdot (b \bmod m)$  关于  $m$  同余，因此可以直接拷贝先前的代码，在每次乘法之后取余即可

```

long long binpow(long long a, long long b, long long m) {
    a %= m;
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}

```

注意：如果  $m$  是一个素数，那么根据 [费马小定理](#)， $x^n$  和  $x^{n \bmod (m-1)}$  是关于  $m$  同余的

## 斐波那契数

问题：求第  $n$  个斐波那契数  $F_n$

做法：本章只简单介绍一下算法的思想，更多细节参考 [斐波那契数](#) 一章。在计算斐波那契数时，我们可以用一个  $2 \times 2$  的矩阵来描述从  $F_i, F_{i+1}$  到  $F_{i+1}, F_{i+2}$  的转换关系。矩阵乘法满足结合律，我们可以用  $O(\log n)$  次操作计算该矩阵的  $n$  次幂

## 施加同一置换k次

问题：给定一个长度为  $n$  的序列和序列上的一个置换，问施加  $k$  次该置换后的结果

做法：先对  $k$  次置换叠加使用快速幂算法得到一个新的置换，然后施加到原序列即可，复杂度  $O(n \log k)$

注意：该问题有线性解法，可以先找到置换中的循环节。分别考虑每个循环节，把  $k$  对循环节的大小取模，接着找到每个数的最终位置

## 坐标系中点的操作

问题：给定三维坐标系中的  $n$  个点  $p_i$ ，对每个点均施加  $m$  次操作，每次操作是平移、伸缩、旋转、循环中的一种，其中循环是将一系列操作重复  $k$  次。如果我们将循环展开，得到总的操作次数是  $length$ ，要求在  $O(n \cdot length)$  的复杂度内完成所有操作

做法：

平移、伸缩（将每维坐标分别乘上一个常数系数）、旋转均是线性操作，循环则是它们的组合，因此所有操作都可以用矩阵描述：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

我们需要对三维坐标系建立一个四维的变换矩阵是因为需要一个常数维度来完成平移操作（每维坐标加上特定常数）。进而，一次操作可以表示为：

$$\begin{Bmatrix} x & y & z & 1 \end{Bmatrix} \begin{Bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{Bmatrix} = \begin{Bmatrix} x' & y' & z' & 1 \end{Bmatrix}$$

当所有操作都被一个矩阵描述后，每种非循环次数都是常数时间完成，而重复  $k$  次的循环操作则是一组矩阵乘积的  $k$  次幂。我们可以预先处理所有操作，时间复杂度为  $O(m \log k)$ ，最后再把总的操作效果施加到  $n$  个点上，总复杂度  $O(n + m \log k)$

## 图上长度为k的路径数

**问题：**给定有  $n$  个顶点的无向无权图，求从顶点  $u$  出发到任意顶点  $v$  的长度为  $k$  的路径数

**做法：**完整做法请参考 [这篇文章](#)，思路是建立该图的邻接矩阵  $M$ ， $M_{ij}$  为 1 代表顶点  $i$  和  $j$  之间有边相连，否则为 0。那么对矩阵  $M$  求  $k$  次幂， $M_{ij}^k$  就代表了从顶点  $i$  到  $j$  的长度为  $k$  的路径数

**注意：**在同一篇文章中还有一个类似问题，找出从顶点  $u$  出发长度为  $k$  的路径中权重最小的路径。大致思想还是通过邻接矩阵的  $k$  次幂解决

## 快速幂变种：大数相乘模m









**问题：**求两个数  $a$  和  $b$  相乘对  $m$  取模的结果， $a, b$  可以用 64-bit 表示，但它们的乘积不可以。额外要求不允许使用大整数

**做法：**类似快速幂，把其中一个数  $a$  拆分成二进制表示，每一位分别与  $b$  相乘。只是快速幂中若干乘积之间还是相乘，现在改为相加，具体见如下递归式：

$$a \cdot b = \begin{cases} 0 & \text{if } a == 0 \\ 2 \cdot \frac{a}{2} \cdot b & \text{if } a > 0 \text{ and } a \text{ even} \\ 2 \cdot \frac{a-1}{2} \cdot b + b & \text{if } a > 0 \text{ and } a \text{ odd} \end{cases}$$

**注意：**[黑科技](#)

## 练习题

-  [UVa 1230 - MODEX](#)
-  [UVa 374 - Big Mod](#)
-  [UVa 11029 - Leading and Trailing](#)
-  [Codeforces - Parking Lot](#)
-  [SPOJ - The last digit](#)
-  [SPOJ - Locker](#)
-  [LA - 3722 Jewel-eating Monsters](#)
-  [SPOJ - Just add it](#)

