[toc]

快速幂，二进制取幂（Binary Exponentiation，也称平方法），是一个在 $\Theta(\log n)$ 的时间内计算 a^n 的小技巧，而暴力的计算需要 $\Theta(n)$ 的时间。而这个技巧也常常用在非计算的场景，因为它可以应用在任何具有结合律的运算中。其中显然的是它可以应用于模意义下取幂、矩阵幂等运算，我们接下来会讨论。

算法描述

计算 a 的 n 次方表示将 n 个 a 乘在一起：$a^{n} = \underbrace{a \times a \cdots \times a}_{n\text{ 个 a}}$。然而当 a,n 太大的时侯，这种方法就不太适用了。不过我们知道：$a^{b+c} = a^b \cdot a^c,,,a^{2b} = a^b \cdot a^b = (a^b)^2$。二进制取幂的想法是，我们将取幂的任务按照指数的 **二进制表示** 来分割成更小的任务。

首先我们将 n 表示为 2 进制，举一个例子：

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

因为 n 有 $\lfloor \log_2 n \rfloor + 1$ 个二进制位，因此当我们知道了 $a^1, a^2, a^4, a^8, \dots, a^{2^{\lfloor \log_2 n \rfloor}}$ 后，我们只用计算 $\Theta(\log n)$ 次乘法就可以计算出 a^n。

于是我们只需要知道一个快速的方法来计算上述 3 的 2^k 次幂的序列。这个问题很简单，因为序列中（除第一个）任意一个元素就是其前一个元素的平方。举一个例子：

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

因此为了计算 3^{13}，我们只需要将对应二进制位为 1 的整系数幂乘起来就行了：

$$ 3^{13} = 6561 \cdot 81 \cdot 3 = 1594323 $$

将上述过程说得形式化一些，如果把 n 写作二进制为 $(n_tn_{t-1}\cdots n_1n_0)_2$，那么有：

$$ n = n_t2^t + n_{t-1}2^{t-1} + n_{t-2}2^{t-2} + \cdots + n_12^1 + n_02^0 $$

其中 $n_i\in{0,1}$。那么就有

$$ \begin{aligned} a^n & = (a^{n_t 2^t + \cdots + n_0 2^0})\\ & = a^{n_0 2^0} \times a^{n_1 2^1}\times \cdots \times a^{n_t2^t} \end{aligned} $$

根据上式我们发现，原问题被我们转化成了形式相同的子问题的乘积，并且我们可以在常数时间内从 2^i 项推出 2^{i+1} 项。

这个算法的复杂度是 $\Theta(\log n)$ 的，我们计算了 $\Theta(\log n)$ 个 2^k 次幂的数，然后花费 $\Theta(\log n)$ 的时间选择二进制为 1 对应的幂来相乘。

代码实现

首先我们可以直接按照上述递归方法实现：

```
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;
}
```

第二种实现方法是非递归式的。它在循环的过程中将二进制位为 1 时对应的幂累乘到答案中。尽管两者的理论复杂度是相同的，但第二种在实践过程中的速度是比第一种更快的，因为递归会花费一定的开销。

```
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;
}
```

模板：Luogu P1226

应用

模意义下取幂

???+note "问题描述" 计算 $x^n\bmod m$。

这是一个非常常见的应用，例如它可以用于计算模意义下的乘法逆元。

既然我们知道取模的运算不会干涉乘法运算，因此我们只需要在计算的过程中取模即可。

```
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：根据费马小定理，如果 m 是一个质数，我们可以计算 $x^{n\bmod (m-1)}$ 来加速算法过程。

计算斐波那契数, 矩阵快速幂

???+note "问题描述" 计算斐波那契数列第 n 项 F_n。

根据斐波那契数列的递推式 $F_n = F_{n-1} + F_{n-2}$，我们可以构建一个 2×2 的矩阵来表示从 F_i,F_{i+1} 到 F_{i+1},F_{i+2} 的变换。于是在计算这个矩阵的 n 次幂的时候，我们使用快速幂的思想，可以在 $\Theta(\log n)$ 的时间内计算出结果。