算法笔记

快速幂

基本原理:

$$a^b = \begin{cases} a \cdot a^{b-1}, & b\text{为奇数} \\ \left(a^{\frac{b}{2}}\right)^2, & b\text{为偶数}. \end{cases}$$

例如求2^{13}的过程如下:

1. $a^{13} = a \times a^{12}$;
2. $a^{12} = \left(a^6\right)^2$;
3. $a^6 = \left(a^3\right)^2$;
4. $a^3 = a \times a^2$;
5. $a^2 = (a)^2$;
6. $a = a \times 1$.
   算法时间复杂度为$O(\log n)$。
   递归实现

```
// 计算 a^b mod m, m为大质数
int binaryPow(int a, int b, int m) {
        if (b == 0) return 1;  // a^0 = 1
        if (b & 1) {  // b & 1 等价于 b % 2 == 1, 判断b的奇偶性
                return a * binaryPow(a, b - 1, m) % m;
        } else {
                int mul = binaryPow(a, b / 2, m);
                return mul * mul % m;
        }
}
```

注意如果 `b` 为偶数时直接 `return binaryPow(a, b/2, m) * binaryPow(a, b/2, m)` 这样时间复杂度仍然是$O(n)$。
考虑上面a^{13}的例子，可以将任意正整数分解为一系列2的幂之和且分解唯一，如 $13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 1101_{(2)}$，因此$a^{13} = a^8 \times a^4 \times a^1$。
迭代实现

```
// 计算 a^b mod m, m为大质数
int binaryPow(int a, int b, int m) {
        int ans = 1;
        while (b > 0) {
                if (b & 1) {  // b的二进制末尾为1
```

```
                ans = ans * a % m;
            }
            a = a * a % m;
            b >>= 1;  // b右移一位
        }
        return ans;
}
```