繰り返し二乗法

繰り返し二乗法

繰り返し二乗法 is an method to calc a^n in $O(\log n)$

- Japanese: Repetitive Squaring Method
- English : Exponentation by Squaring

Method

1. Recursive Method
2. Iterative Method

Latter is better for memory.

Recursive Method

Use this observation:

$$a^n = \begin{cases} a \cdot a^{\frac{n}{2}} a^{\frac{n}{2}} & n : odd \\ a^{\frac{n}{2}} a^{\frac{n}{2}} & n : even \end{cases}$$

$\frac{n}{2}$: **Floor** of n devided 2

Example

Go down to 3^0 (See up to down.)

$$
\begin{aligned}
3^{10} &= 3^5 3^5 \\
3^5 &= 3 \cdot 3^2 3^2 \\
3^2 &= 3^1 3^1 \\
3^1 &= 3 \cdot 3^0 3^0 \\
3^0 &= 1
\end{aligned}
$$

Return to 3^{10} (See down to up.)

$$3^{10} = 243 \cdot 243 = 59409$$

$$3^{5} = 3 \cdot 9 \cdot 9 = 243$$

$$3^{2} = 3 \cdot 3 = 9$$

$$3^{1} = 3 \cdot 1 \cdot 1 = 3$$

$$3^{0} = 1$$

AOJ NTL_1_B - Power

```
#define MOD 1000000007
long long modpow(long long a, long long n)
{
  if (n == 0) return 1;
  long long b = modpow(a, n/2);
  b = b*b % MOD;
  if (n % 2 == 0) return b;
  else return a*b % MOD;
}
```

- You should use modulo operation for each calculation for overflow.
- Calculation of three elements may cause overflow.

  ex: a*b*b causes overflow.

Iterative Method

Use this observation:

n is expressed as **binary number**

$$a^n = a^{b_m 2^m + b_{m-1} 2^{m-1} + \cdots + b_0 2^0}$$

$$b_k = 0 \; or \; 1$$

Example

$$
\begin{aligned}
3^{10} &= 3^{(1010)_2} \\
&= 3^{1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0} \\
&= 3^{1\cdot 2^3}\, 3^{0\cdot 2^2}\, 3^{1\cdot 2^1}\, 3^{0\cdot 2^0} \\
&= 3^{1\cdot 8}\, 3^{0\cdot 4}\, 3^{1\cdot 2}\, 3^{0\cdot 1} \\
&= 3^{1\cdot 8}\, 3^{1\cdot 2}
\end{aligned}
$$

$$(1010)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

$$ans := 1$$

$$3^1 = 3 \qquad \Rightarrow ans = 1$$

$$3^2 = 3^1 3^1 \quad \Rightarrow ans = 3^2$$

$$3^4 = 3^2 3^2 \quad \Rightarrow ans = 3^2$$

$$3^8 = 3^4 3^4 \quad \Rightarrow ans = 3^8 3^2$$

```
#define MOD 1000000007
long long modpow(long long a, long long n)
{
  long long ret = 1;
  while (n > 0) {
    if (n % 2 == 1) ret *= a;
    ret %= MOD;
    a = a*a % MOD;
    n /= 2;
  }
  return ret;
}
```

Application

- power of matrix
- modulo operation for division
    - fast calculation of combination