二分快速幂

二分法 第二课

重庆南开信竞基础课程 Helang

$a^n \% c = ?$ $0<=n<=1,000,000,000$

算法1.首先直接地来设计这个算法:

```
int ans = 1;
for(int i = 1; i<=n; i++) ans = ans * a;
ans = ans % c;
```

这个算法的时间复杂度体现在for循环中，时间复杂度为$O(n)$
这个算法存在着明显的问题，如果a和n过大，很容易就会溢出。

第一个改进方案，有这样一个公式：

公式1: $(a*b) \% c == ((a \% c) * (b \% c)) \% c$

$(a*b) \% c == ((a \% c) * b) \% c$

$(a^n)\%c == (a*a*......*a)\%c$

$== ((a\%c)*(a\%c)*......*(a\%c))\%c == ((a\%c)^n)\%c$

结论：$a^n \% c == (a\%c)^n \% c$

```
//改进算法1，得到算法2：
int ans = 1;
a = a % c;  //加上这一句
for(int i = 1;i<=n;i++)ans = ans * a;
ans = ans % c;
```

```
//上一页的算法1：
int ans = 1;
for(int i = 1;i<=b;i++)ans = ans * a;
ans = ans % c;
```

公式1: (a*b) % c ==((a % c)*(b % c)) %c

(a*b) % c == ((a % c) * b) % c

既然相乘再取余与先取余再相乘再取余保持余数不变，那么新算得的ans也可以进行取余，所以可以再次改进算法。

算法2:
```
int ans = 1;
a = a % c;
for(int i = 1; i<=n; i++)ans = ans * a;
ans = ans % c;  //ans先做乘法，再取余
```

算法3:
```
int ans = 1;
a = a % c;
for(int i = 1; i<=n; i++)ans = (ans * a) % c;//这里再取了一次余
ans = ans % c;
```

这个算法在时间复杂度上没有改进，仍为 $O(n)$ ，不过已经好很多了，至少不会溢出了。但是在n过大的条件下，还是很有可能超时。

公式2: $a^n \% c == (a\%c)^n \% c$

$a^n\%c==((a^2)^{n/2})\%c == ((a^2 \% c)^{n/2})\%c$ n为偶数

$a^n\%c==(a*(a^2)^{n/2})\%c == (a*(a^2 \% c)^{n/2})\%c$ n为奇数

有了上述两个公式后，我们可以得出以下的结论：
1.如果n是偶数，我们可以记$b = a^2 \% c$，那么求$b^{n/2} \% c$ 就可以了。
2.如果n是奇数，我们可以记$b = a^2 \% c$，那么求$(b^{n/2} \% c * a) \% c$ 就可以了。

根据上述结论，怎样改进算法3?

```
int ans = 1;
a = a % c;
for(int i = 1;i<=n;i++)ans = (ans * a) % c;
ans = ans % c;
```

那么我们可以得到以下算法：
算法4：

```
int ans = 1;
a = a % c;
b = (a*a) % c;                      //相当于用b代替了a²
if(n%2==1)ans = (ans * a) % c;  //如果是奇数，要多求一步，可以提前算到ans中
for(int i = 1;i<=n/2;i++) ans = (ans * b) % c;
ans = ans % c;
```

这样我们把时间复杂度优化成了O(n/2)

我们把时间复杂度变成了O(n/2).当然，这样子治标不治本。

但我们看到，当我们令b = (a * a)% c时，状态已经发生了变化，我们要求的最终结果即为 $(b)^{n/2}$ % c而不是原来的 a^n % c，这个过程是可以迭代下去的。

下一步再令 **x=(b*b)%c** 偶数就是求 **$x^{n/4}$ % c** 奇数就是 **($x^{n/4}$ % c * a)% c**

再下一步令 **y=(x*x)%c** 偶数就是求 **$y^{n/8}$ % c** 奇数就是 **($y^{n/8}$ % c * a)% c**

…… 直到**指数n/△==0**

当然，对于**奇数**的情形会多出一项a % c，所以为了完成迭代，当n是奇数时，我们通过ans = (ans * a) % c; 来弥**补**多出来的这一项。

形如上式的迭代下去后，每次把指数n=n/2; 当n==0时，所有的因子都已经相乘，算法结束。于是便可以在$O(logn)$ 的时间内完成了。于是，有了最终的算法：快速幂算法。

```
int ans = 1;
a = a % c;
while(n>0)
{
        if(n % 2 == 1)ans = (ans * a) % c;    //奇数，补一项
        n = n/2;
        a = (a * a) % c;
}
cout<<ans<<endl;
```

蒙哥马利快速幂取模算法，简单漂亮

```
int Montgomery(int a,int n,int c) //求aⁿ % c
{
        int ans=1;
        a=a%c;
        while(n>0)
        {
                if(n%2==1)ans=(ans*a)%c;      //奇数，补一项，
                n=n/2;
                a=(a*a)%c;
        }
        return ans;
}
```

Montgomery reduction由**彼得·蒙哥马利**在1985年提出。
时间复杂度$O(\log_2 n)$

经典题目选讲NKOJ 2493

n个小伙伴（编号从0到n-1）围坐一圈玩游戏。按照顺时针方向给n个位置编号，从0到n-1。最初，第0号小伙伴在第0号位置，第1号小伙伴在第 1 号位置……，依此类推。

游戏规则如下：每一轮第0号位置上的小伙伴顺时针走到第m号位置，第1号位置小伙伴走到第m+1号位置……，依此类推，第n-m号位置上的小伙伴走到第0号位置，第n-m+1号位置上的小伙伴走到第1号位置……，第n-1号位置上的小伙伴顺时针走到第m-1号位置。

现在，一共进行了10^k轮，请问x号小伙伴最后走到了第几号位置。

数据规模：$2<=n<=10^6, 1<=m<=n, 1<=x<=n, 1<=k<=10^9$.

经典题目选讲NKOJ 2493

我们从简单的0号小伙伴讨论起：

每次移动m步，移动10^k次后，0号小伙伴总共移动了10^k*m步，此时他所在的位置编号为？

$$(10^k*m)\%n$$

此时，x号小伙伴所在的位置为?

$$(10^k*m)\%n+x$$

但$(10^k*m)\%n+x$可能超过n的范围，所以再模一次，变为$((10^k*m)\%n+x)\%n$，**这就是所求的答案！**

现在的问题成了快速求解$(10^k*m)\%n$

根据前面所学公式

$$(a*b)\ \%\ c == (\ (a\ \%\ c)*(b\ \%\ c)\ )\ \%\ c$$

$$(a*b)\ \%\ c == (\ (a\ \%\ c)\ *\ b\ )\ \%\ c$$

$(10^k*m)\%n==((10^k\%n)*m)\%n$,用快速幂求出$10^k\%n$即可！

最终答案式子：$(((10^k\%n)*m)\%n+x)\%n$

```
long long Montgomery(long long a,long long b,long long c)
{
        long long ans=1;
        a=a%c;
        while(b>0)
        {
                if(b%2==1)ans=(ans*a)%c;
                b=b/2;
                a=(a*a)%c;
        }
        return ans;
}
int main()
{
        long long n,m,x,k,tmp,Result;
        scanf("%lld%lld%lld%lld",&n,&m,&k,&x);
        tmp=Montgomery(10,k,n);    //求10^k%n
        Result=((tmp*m)%n+x)%n;    //计算(((10^k%n)*m)%n+x)%n
        printf("%lld\n",Result);
}
```