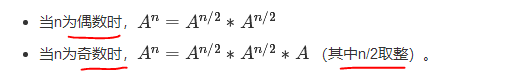


由于矩阵乘法具有结合律, 因此



我们可以得到这样的结论:



这就告诉我们, 计算A^n也可以使用二分快速求幂的方法!!!

例如, 为了算出A^25的值, 我们只需要递归地计算出A^12、A^6、A^3的值即可。

根据这里的一些结果, 我们可以在计算过程中不断取模, 避免高精度运算。

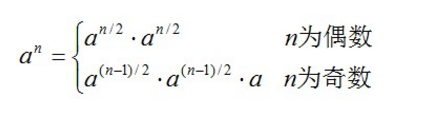
**快速求幂取模**

公式求幂→二分求幂→快速求幂→快速求幂取模

等不急的可以直接下拉到最后看快速幂取模。

直接用C语言的库函数pow()（别忘了它的头文件#include<math.h>），似乎很简单，但是它的时间复杂度高达O(n)。   
显然，这很容易超时。   
于是有了下面的二分求幂（时间复杂度O(lgn)）

二分求幂的原理可以用下面这张图表示



用递归来实现，虽然代码有点长，但是很好理解

int pow(int a,int n)//返回值是a的n次方

{

if(n==0)//递归终止条件

return 1;

if(n==1)

return a;

int result=pow(a,n/2);//二分递归

result=result\*result;//这部分奇数偶数都一样

if(n%2==1)//如果n是奇数，就要多乘一次

result=result\*a;

return result;

}

用非递归，更加简洁

int pow(int a,int n)//返回值是a的n次方

{

int result=1;

while(n!=0)

{

if(n%2==1)//如果n是奇数

result=result\*a;//就要多乘一次

a=a\*a;

n=n/2;//二分

}

return result;

}

快速幂顾名思义比二分幂又快一些，   
快速幂借助了强大的位运算，时间复杂度达到O(log₂N)。   
位运算<http://baike.baidu.com/view/379209.htm>   
原理其实不难理解，用的还是小学生的公式。   
例如 求a的11次方   
11的二进制是1011   
11 = 2³×1 + 2²×0 + 2¹×1 + 2º×1

因此，我们将a¹¹转化为算

用非递归的代码实现

int pow(int a,int n)//返回值是a的n次方

{

int result=1,flag=a;

while(n!=0)

{

if(n&1)//如果n是奇数，即n的二进制最末位为1时

result=result\*flag;

flag=flag\*flag;

n=n>>1;//n的二进制右移一位，即n/2

}

return result;

}

当然还能用递归来实现，但是太复杂，我没学会…

刷题中让直接求幂的不多，求幂后取模的却不少，毕竟求幂结果太大了。   
水平所限，只会用二分幂取模，时间复杂度与二分幂一样O(lgn)。   
基本可以在各种比赛中顺利通过，也是目前比较常用的方法

原理同样很简单，都是小学学过的：积的取余等于取余的积取余   
接下来用代码实现

int pow(int a,int n,int b)//返回值是a的n次方对b取余后的值

{

int result=1;

a=a%b;//积的取余等于取余的积取余

while(n>0)

{

if(n%2==1)

result=result\*a%b;//n是奇数的话就要多乘一次，原理和前面的二分求幂一样

n=n/2;//二分

a=a\*a%b;//积的取余等于取余的积取余

}

return result;

}

影响计算机效率的是运算次数，而不是运算结果。   
所以前面几个算法都是通过增大运算结果，减少运算次数，提高计算机效率。