

# MAXimal

[home](#)
[algo](#)
[bookz](#)
[forum](#)
[about](#)

added: 10 Jun 2008 16:41

Edited: 3 May 2012 2:34

## Binary exponentiation

Binary (binary) exponentiation - is a technique that allows you to build any number  $n$  of the degree of the  $O(\log n)$  multiplications (instead of  $n$  the usual approach of multiplications).

Moreover, the technique described here is applicable to any of the **associative** operation, and not only to the multiplication number. Recall operation is called associative if for any  $a, b, c$  executed:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

The most obvious generalization - on residues modulo some (obviously, associativity is maintained). The next "popularity" is a generalization to the matrix product (it is well-known associativity).

## Algorithm

Note that for any number  $a$  and an **even** number of  $n$  feasible obvious identity (which follows from the associativity of multiplication):

$$a^n = (a^{n/2})^2 = a^{n/2} \cdot a^{n/2}$$

It is the main method in the binary exponentiation. Indeed, for even  $n$  we have shown how, by spending only one multiplication, one can reduce the problem to less than half the power.

It remains to understand what to do if the degree is **odd**. Here we do is very simple: move on to the power that will have an even:  $n - 1$

$$a^n = a^{n-1} \cdot a$$

So, we actually found a recursive formula on the degree  $n$  we go, if it even, to  $n/2$ , and otherwise - to  $n - 1$ . understandable that there will be no more  $2 \log n$  hits before we come to  $n = 0$  (the base of the recursion formula). Thus, we have

### Contents [hide]

- Binary exponentiation
  - Algorithm
  - Implementation
  - Examples of solving problems
    - Efficient calculation of Fibonacci numbers
    - Erection of changes in the  $k$  degree of th
    - The rapid deployment of a set of geometric operations to points
    - Number of tracks fixed length column
    - Variation binary exponentiation: the multiplication of two numbers modulo
  - Problem in online judges

an algorithm that works for  $O(\log n)$  multiplications.

## Implementation

A simple recursive implementation:

```
int binpow (int a, int n) {
    if (n == 0)
        return 1;
    if (n % 2 == 1)
        return binpow (a, n-1) * a;
    else {
        int b = binpow (a, n/2);
        return b * b;
    }
}
```

Non-recursive implementation, also optimized (division by 2 replaced bit operations):

```
int binpow (int a, int n) {
    int res = 1;
    while (n)
        if (n & 1) {
            res *= a;
            --n;
        }
        else {
            a *= a;
            n >>= 1;
        }
    return res;
}
```

This implementation is still somewhat simplified by noting that the construction of *a* the square is always performed, regardless of whether the condition is odd worked *nor* not:

```
int binpow (int a, int n) {
    int res = 1;
    while (n) {
        if (n & 1)
            res *= a;

        a *= a;
        n >>= 1;
    }
    return res;
}
```

Finally, it is worth noting that the binary exponentiation already implemented by

Java, but only for a class of long arithmetic BigInteger (pow function in this class works under binary exponentiation algorithm).

## Examples of solving problems

### Efficient calculation of Fibonacci numbers

**Condition** . Given a number  $n$ . Required to calculate  $F_n$  where  $F_i$ - the Fibonacci sequence .

**Decision** . More detail the decision described in [the article on the Fibonacci sequence](#) . Here we only briefly we present the essence of the decision.

The basic idea is as follows. Calculation of the next Fibonacci number is based on the knowledge of the previous two Fibonacci numbers, namely, each successive Fibonacci number is the sum of the previous two. This means that we can construct a matrix  $2 \times 2$  that will fit this transformation: as the two Fibonacci numbers  $F_i$  and  $F_{i+1}$  calculate the next number, ie pass to the pair  $F_{i+1}, F_{i+2}$ . For example, applying this transformation  $n$  to a couple times  $F_0$  and  $F_1$  we get a couple  $F_n$  and  $F_{n+1}$ . Thus, elevating the matrix of this transformation in  $n$  the power of  $th$ , we thus find the required  $F_n$  time for  $O(\log n)$  what we required.

### Erection of changes in the $k$ degree of $th$

**Condition** . Given permutation of  $p$  length  $n$ . Required to build it in  $k$  the power of  $th$ , ie find what happens if to the identity permutation  $k$  permutation times apply  $p$ .

**Decision** . Just apply to the interchange of  $p$  the algorithm described above binary exponentiation. No differences compared with the construction of the power of numbers - no. Solution is obtained with the asymptotic behavior  $O(n \log k)$ .

(Note. This problem can be solved more efficiently, **in linear time** . To do this, just select all the cycles in the permutation, and then to consider separately each cycle, and taking  $k$  modulo the length of the current cycle, to find an answer for this cycle.)

### The rapid deployment of a set of geometric operations to points

**Condition** . Given  $n$  points  $p_i$  and are  $m$  transformations that must be applied to each of these points. Each transformation - a shift or a predetermined vector, or scaling (multiplication coefficients set of coordinates) or a rotation around a predetermined axis at a predetermined angle. In addition, there is a composite operation is cyclical repetition: it has a kind of "repeat a specified number of times specified by the list of transformations" (cyclic repetition of operations can be nested).

Required to compute the result of applying these operations to all points

(effectively, ie in less than what  $O(n \cdot \text{length})$ , where *length*- the total number of operations that must be done).

**Decision** . Look at the different types of transformations in terms of how they change location:

- Shift operation - it just adds to all the coordinates of the unit, multiplied by some constant.
- Zoom operation - it multiplies each coordinate by a constant.
- Operation rotational axis - it can be represented as follows: new received coordinates can be written as a linear combination of the old.

(We will not specify how this is done. Example, you can submit it for simplicity as a combination of five-dimensional rotations: first, in the planes  $OXY$ , and  $OXZ$  so that the axis of rotation coincides with the positive direction of the axis  $OX$ , then the desired rotation around an axis in the plane  $YZ$ , then reverse rotations in planes  $OXZ$  and  $OXY$  so that the axis of rotation back to its starting position.)

Easy to see that each of these transformations - a recalculation of coordinates on linear formulas. Thus, any such transformation can be written in matrix form  $4 \times 4$  :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix},$$

which, when multiplied (left) to the line of the old coordinates and constant unit gives a string of new coordinates and constants units:

$$(x \quad y \quad z \quad 1) \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = (x' \quad y' \quad z' \quad 1).$$

(Why it took to enter a dummy fourth coordinate that is always equal to one? Without this we would have to implement the shift operation: after the shift - it is just adding to the coordinates of the unit, multiplied by certain coefficients. Without dummy units we would be able to implement only linear combinations of the coordinates themselves, and add thereto predetermined constant - could not.)

Now the solution of the problem becomes almost trivial. Times each elementary operation described by the matrix, then the process described by the product of these matrices, and the operation of a cyclic repetition - the erection of this matrix to a power. Thus, during the time we  $O(m \cdot \log \text{repetition})$  can predposchitat matrix  $4 \times 4$  that describes all the changes, and then just multiply each point  $p_i$  on the matrix - thus, we will answer all questions in time  $O(n)$ .

## Number of tracks fixed length column

**Condition** . Given an undirected graph  $G$  with  $n$  vertices, and given a number  $k$ . Required for each pair of vertices  $i$ , and  $j$  find the number of paths between them containing exactly  $k$  the edges.

**Decision** . More details on this problem is considered in [a separate article](#) . Here, we only recall the essence of this solution: we simply erect a  $k$  degree of the adjacency matrix of the graph, and the matrix elements, and will be the solutions. Total asymptotics -  $O(n^3 \log k)$ .

(Note. In [the same article](#), and the other is considered a variation of this problem: when the graph is weighted, and you want to find the path of minimum weight, containing exactly  $k$  the edges. As shown in this paper, this problem is also solved by using binary exponentiation of the adjacency matrix of the graph, but instead of the usual operation of multiplication of two matrices should be used modified: instead of multiplying the amount taken, and instead of summing - Take a minimum.)


## Variation binary exponentiation: the multiplication of two numbers modulo

We give here an interesting variation of the binary exponentiation.

Suppose we are faced with such a **task** : to multiply two numbers  $a$  and  $b$  modulo  $m$ :

$$a \cdot b \pmod{m}$$

Assume that the numbers can be quite large: so that the numbers themselves are placed in a built-in data types, but their direct product  $a \cdot b$  - no longer exists (note that we also need to sum numbers placed in a built-in data type). Accordingly, the task is to find the desired value  $(a \cdot b) \pmod{m}$ , without the aid of [a long arithmetic](#) .

**The decision** is as follows. We simply apply the binary exponentiation algorithm described above, but instead of multiplication, we will make the addition. In other words, the multiplication of two numbers, we have reduced to  the operations of addition and multiplication by two (which is also, in fact, is the addition).

(Note. This problem can be solved **in a different way** , by resorting to assistance operations with floating-point numbers. Namely, to calculate the floating point expression  $a \cdot b / m$ , and round it to the nearest integer. So we find **an approximate** quotient. Subtracting from his works  $a \cdot b$  (ignoring overflow), we are likely to get a relatively small number, which can be taken modulo  $m$  - and return it as an answer. This solution looks pretty unreliable, but it is very fast, and very briefly, is realized.)

## Problem in online judges

List of tasks that can be solved using binary exponentiation:

- [SGU # 265 "Wizards"](#) [Difficulty: Medium]
-

5 Комментариев

e-maxx

 Войти ▾

Лучшее вначале ▾

Поделиться  Избранный ★

Присоединиться к обсуждению...



Петр • 9 месяцев назад

можно увидеть пример реализации быстрого возведения в степень по модулю?

3 ^ | ▾ • Ответить • Поделиться ›

**Константин Ольмезов** → Петр • 7 месяцев назад

```
int pow_mod(int a,int b,int m)
{
    int r=1;
    a%=m;
    while (b)
    {
        if (b&1)
            r=(r*a)%m;
        a=(a*a)%m;
        b>>=1;
    }
    return r;
}
```

2 ^ | ▾ • Ответить • Поделиться ›

**Ramil6085** • 2 года назад

вопрос по пункту "перемножение двух чисел по модулю". Мне не очень понятно, откуда взялась асимптотика  $\log(m)$ . Возможно, имелось ввиду  $\log(\max(a, b))$ ?

1 ^ | ▾ • Ответить • Поделиться ›

**e\_maxx** Модератор → Ramil6085 • 2 года назад

$O(\log(m))$  - это я взял оценку сверху. На самом деле, да, даже такую асимптотику можно написать:  $O(\log(\min(a, b)))$  - потому что можно идти по битам меньшего числа, производя сложения над большим числом.

1 ^ | ▾ • Ответить • Поделиться ›

**unknown** • 2 года назад $(a^b) \% c \rightarrow 1 \leq a \leq b \leq 10^{18}$