A tutorial on Fast Modulo Multiplication (Exponential Squaring)

48

44

Hello @all,

This tutorial will cover the topic of Fast Modulo Multiplication (also known as Exponential Squaring or Repeated Squaring).

This is a very useful technique to have under your arsenal as a competitive programmer, especially because such technique often appears on Maths related problems and, sometimes, it might be the difference between having AC veredict or TLE veredict, so, for someone who still doesn't know about it, I hope this tutorial will help :)

- **Main reason why the usage of repeated squaring is useful**

This technique might seem a bit too complicated at a first glance for a newbie, after all, say, we want to compute the number 3^{10}. We can simply write the following code and do a simple for loop:

```
#include <iostream>
using namespace std;

int main()
{
    int base = 3;
    int exp = 10;
    int ans = 1;
    for(int i = 1; i <= exp; i++)
    {
        ans *= base;
    }
    cout << ans;
    return 0;
}
```

The above code will correctly compute the desired number, 59049. However, after we analyze it carefully, we will obtain an insight which will be the key to develop a faster method.

Apart from being correct, the main issue with the above method is **the number of multiplications that are being executed**.

We see that this method executes exactly **exp-1** multiplications to compute the number n^{exp}, which is not desirable when the value of **exp** is very large.

Usually, on contest problems, the idea of computing large powers of a number appears coupled with the existence of a modulus value, i.e., as the values being computed can get very large, very fast, the value we are looking to compute is usually something of the form:

$n^{exp} \% M$,

where M is usually a large prime number (typically, $10^9 + 7$).

Note that we could still use the modulus in our naive way of computing a large power: we simply use modulus on all the intermediate steps and take modulus at the end, to ensure every calculation is kept within the limit of "safe" data-types.

- **The fast-exponentiation method: an implementation in C++**

It is possible to find several formulas and several ways of performing fast exponentiation by searching over the internet, but, there's nothing like implementing them on our own to get a better feel about what we are doing :)

I will describe here the most naive method of performing repeated squaring. As found on Wikipedia, the main formula is:

$$x^n = \begin{cases} x\,(x^2)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\ (x^2)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

A brief analysis of this formula (an intuitive analysis if you prefer), based both on its recursive formulation and on its implementation allows us to see that the formula uses only $O(\log_2 n)$ squarings and $O(\log_2 n)$ multiplications!

This is a major improvement over the most naive method described in the beginning of this text, where we used much more multiplication operations.

Below, I provide a code which computes $base^{exp} \%$ 1000000007, based on wikipedia formula:

```
{
    if(exp==1)
    return base;
    else
    {
        if(exp%2 == 0)
        {
            long long int base1 = pow(fast_exp(base, exp/2),2);
            if(base1 >= 1000000007)
            return base1%1000000007;
            else
            return base1;
        }
        else
        {
            long long int ans = (base*  pow(fast_exp(base,(exp-1)/2),2));
            if(ans >= 1000000007)
            return ans%1000000007;
            else
            return ans;
        }
    }
}
```

- **The 2^k-ary method for repeated squaring**

Besides the recursive method detailed above, we can use yet another insight which allows us to compute the value of the desired power.

The main idea is that we can expand the exponent in base 2 (or more generally, in base 2^k, with k >= 1) and use this expansion to achieve the same result as above, but, this time, using less memory.

Below you can find the code provided in the comment by @michal27:

```
ll fast_exp(int base, int exp) {
    ll res=1;
    while(exp>0) {
       if(exp%2==1) res=(res*base)%MOD;
       base=(base*base)%MOD;
       exp/=2;
    }
    return res%MOD;
}
```

These are the two most common ways of performing repeated squaring on a live contest and I hope you have enjoyed this text and have found it useful :)

- **Useful problems to practice this new concept:**

Kisses & Hugs;

Chef and segments;

- **Further analysis and generalizations**

Besides the simple method explained here, applied only to the computation of large powers of a given number, it turns out that this method is widely used in the fast computation of powers of matrices and, as we shall see on a later tutorial (hopefully, not so later ;)) this is where this method actually excels and provides really good methods for tackling hard problems :)

Best regards,

Bruno

maths tutorial exponentiation

edited 12 Aug '13, 23:33

asked 12 Aug '13, 19:59

2★ kuruma
[17.6k]●72●143●209
accept rate: 8%

1 very well explained bro...it makes a 10/10 tutorial...:)

4★ kunal361 (12 Aug '13, 23:01)

1 I'm really glad you liked it :D

2★ kuruma (12 Aug '13, 23:13)

actually this tutorial is awesome....I m stucked how can any1 explain so simple...I want to learn programming , algorithm in your guidence please provide me some source to read your blog, tutorial any material produced by you or any1 like you.... thanks for appreaciation...

5★ rcsldav2017 (04 Jul '15, 03:30)

16 Answers: oldest answers newest answers popular answers

1 2 next »

21

complexity. Your program has O(log n) memory complexity because of recursion.

Here I use the fact that I can divide exp to sum of powers of two. So the base^exp can be expressed as multiple of base^poweroftwo. So I just in one while cycle decompose exp as binary number and in base I create appropriate power of base (just square of previous base).

In the code, ll stands for long long int. And MOD is choosed modulo.

```
ll fast_exp(int base, int exp) {
    ll res=1;
    while(exp>0) {
       if(exp%2==1) res=(res*base)%MOD;
       base=(base*base)%MOD;
       exp/=2;
    }
    return res;
}
```

link

answered **12 Aug '13, 20:51**
5★ michal27
[1.1k]●2●10●17
accept rate: 13%

Thanks added this to the tutorial :D

2★ kuruma (12 Aug '13, 22:14)

can somebody explain this solution

2★ shreyasnn1 (11 Dec '16, 16:05)

1

Very nice @Kuruma ,also key to problem CHMOD asked in recent August 2013 long challenge,thanks for the tutorial :)

link

answered **12 Aug '13, 20:15**

1★ faiz
[85]●3
accept rate: 0%

good post i like it

1★ driftking (12 Aug '13, 23:59)

I am really glad to hear it, @driftking

2★ kuruma (13 Aug '13, 00:06)

1

I got AC in kisses and hugs only after converting base into long long int

maybe the statement

```
base=(base*base)%MOD;
```

was the cause of overflow.

link

answered **30 Mar '14, 02:04**

4★ grv95
[9]●1●2
accept rate: 0%

0

return should be res%MOD; otherwise (x ^0)%1 will return 1

link

answered **12 Aug '13, 23:30**

5★ moinul_1005015
[1]
accept rate: 0%

1 you are right, fixed it

2★ kuruma (12 Aug '13, 23:35)

in majority, you don't use 1 as modulo but yes, you are right :)

5★ michal27 (12 Aug '13, 23:55)

0

This may give wrong results (for example, for 999,999,999 and 1000,000,000) because of pow() function which uses double. Instead of pow(), for squaring, if we simply multiply the numbers, we will get correct result i.e. 570312504, 140625001. link text

link

edited **30 Jul '14, 15:30**

answered **30 Jul '14, 15:28**
4★ nishant2002
[82]●2●4●7
accept rate: 25%

0

when it comes to modulo of 10^9 it gives some bogus solution 504271345 instead of 4 .
https://www.hackerrank.com/challenges/k-candy-store/submissions/code/10592914

link

answered **20 Dec '14, 22:00**
ism_syed
[1]
accept rate: 0%

3★ betlista ♦♦ (20 Dec '14, 23:09)

Hey,i am not getting how your recursion takes %(100000007) at every recursive step to avoid overflow..Can you explain this little bit.I mean to say how recursion work here?Does it calculate a^b and then calculate %(100000007) or it calculates ((a1%(100000007)*(a2%100000007)...)%(100000007).If the answer is latter can you please explain how

2★ doodle90 (06 Mar '15, 01:05)

0 I have small doubt regarding fast_exp(int base, int exp) function. I think it may return wrong answer in case exp is 1. As for that case the modulo operation is not happening. Can anyone explain?

link

answered **02 Jan '16, 22:50**
aniket153
[1]
accept rate: 0%

0 Hey Guys, just a small note who may be having problems using the above functions. For large values of base and exp you should prefer long long int as data types of base and exponent in fn. Also, second recursive function is better. IF you use int data types you may run into run time error, overflow.

link

edited **03 Apr '16, 16:06**

answered **03 Apr '16, 16:03**

5★ kush2327
[11]●2
accept rate: 0%

0 Great tutorial .... Just a heads-up ... the first method caused a RunTime error (SIGSEGV) . I think that it was caused because of the fact that it is a recursive approach and that it takes up too much of memory . The second method is much better .... but you should initialize both the exp and base as long long int

link

answered **10 Apr '16, 16:42**
2★ codwotin
[1]
accept rate: 0%

0 You can make it even faster by using bitwise operators to check for parity (n&1 instead of n%2==1) and to divide by two (n=n>>1 instead of n=n/2). :)

link

answered **11 Apr '16, 22:20**

4★ mightymercado
[282]●6
accept rate: 11%

1 2 next »

[hide preview] community wiki:

Preview

Post Your Answer