pajenegod's blog

Montgomery Multiplication Explained (Fast Modular Multiplication)

By **pajenegod**, history, 12 months ago,

Hi CF! During this past weekend I was reading up on Montgomery transformation, which is a really interesting and useful technique to do fast modular multiplication. However, all of the explanations I could find online felt very unintuitive for me, so I decided to write my own blog on the subject. A big thanks to **kostia244**, **nor**, **nskybytskyi** and **-is-this-fft-** for reading this blog and giving me some feedback =).

Fast modular multiplication

Let $P = 10^9 + 7$ and let a and b be two numbers in $[0, P)$. Our goal is to calculate $a \cdot b \,\%\, P$ without ever actually calling $\%\, P$. This is because calling $\%\, P$ is very costly.

If you haven't noticed that calling $\%\, P$ is really slow, then the reason you haven't noticed it is likely because the compiler automatically optimizes away the $\%\, P$ call if P is known at compile time. But if P is not known at compile time, then the compiler will have to call $\%\, P$, which is really really slow.

Montgomery reduction of $a \cdot b$

It turns out that the trick to calculate $a \cdot b \,\%\, P$ efficently is to calculate $a \cdot b \cdot 2^{-32} \,\%\, P$ efficiently. So the goal for this section will be to figure out how to calculate $a \cdot b \cdot 2^{-32} \,\%\, P$ efficently. $a \cdot b \cdot 2^{-32} \,\%\, P$ is called the Montgomery reduction of $a \cdot b$, denoted by $\text{m\_reduce}(a \cdot b)$.

Idea (easy case)

Suppose that $a \cdot b$ just happens to be divisible by 2^{32}. Then $(a \cdot b \cdot 2^{-32}) \,\%\, P = (a \cdot b) \gg 32$, which runs super fast!

Idea (general case)

Can we do something similar if $a \cdot b$ is not divisible by 2^{32}? The answer is yes! The trick is to find some integer m such that $(a \cdot b + m \cdot P)$ is divisible by 2^{32}. Then $a \cdot b \cdot 2^{-32} \,\%\, P = (a \cdot b + m \cdot P) \cdot 2^{-32} \,\%\, P = (a \cdot b + m \cdot P) \gg 32$.

So how do we find such an integer m? We want $(a \cdot b + m \cdot P) \,\%\, 2^{32} = 0$ so $m = (-a \cdot b \cdot P^{-1}) \,\%\, 2^{32}$. So if we precalculate $(-P^{-1}) \,\%\, 2^{32}$ then calculating m can be done blazingly fast.

Montgomery transformation

Since the Montgomery reduction divides $a \cdot b$ by 2^{32}, we would like some some way of multiplying by 2^{32} modulo P. The operation $x \cdot 2^{32} \,\%\, P$ is called the Montgomery transform

→ Pay attention

Before contest
Codeforces Round 876 (Div. 2)
08:20:36
Register now »
*has extra registration

→ Top rated

#	User	Rating
1	Benq	3783
2	jiangly	3741
3	tourist	3622
4	Um_nik	3536
5	maroonrk	3526
6	inaFSTream	3477
7	fantasy	3468
8	ecnerwala	3454
9	QAQAutoMaton	3428
10	fivedemands	3381

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Um_nik	184
2	adamant	178
3	awoo	177
4	nor	169
5	maroonrk	165
6	-is-this-fft-	164
7	antontrygubO_o	152
8	ko_osaga	151
9	dario2994	150
10	SecondThread	149

View all →

→ Find user

Handle:

Find

→ Recent actions

awoo → Educational Codeforces Round 26 Editorial

of x, denoted by $\text{m\_transform}(x)$.

The trick to implement m_transform efficently is to make use of the Montgomery reduction. Note that $\text{m\_transform}(x) = \text{m\_reduce}(x \cdot (2^{64} \,\%\, P))$, so if we precalculate $2^{64} \,\%\, P$, then m_transform also runs blazingly fast.

Montgomery multiplication

Using m_reduce and m_transform there are multiple different ways of calculating $a \cdot b \,\%\, P$ effectively. One way is to run $\text{m\_transform}(\text{m\_reduce}(a \cdot b))$. This results in two calls to m_reduce per multiplication.

Another common way to do it is to always keep all integers transformed in the so called Montgomery space. If $a' = \text{m\_transform}(a)$ and $b' = \text{m\_transform}(b)$ then $\text{m\_transform}(a \cdot b \,\%\, P) = \text{m\_reduce}(a' \cdot b')$. This effectively results in one call to m_reduce per multiplication, however you now have to pay to move integers in to and out of the Montgomery space.

Example implementation

Here is a Python 3.8 implementation of Montgomery multiplication. This implementation is just meant to serve as a basic example. Implement it in C++ if you want it to run fast.

```
P = 10**9 + 7
r = 2**32
r2 = r * r % P
Pinv = pow(-P, -1, r) # (-P^-1) % r

def m_reduce(ab):
  m = ab * Pinv % r
  return (ab + m * P) // r

def m_transform(a):
  return m_reduce(a * r2)

# Example of how to use it
a = 123456789
b = 35
a_prim = m_transform(a) # mult a by 2^32
b_prim = m_transform(b) # mult b by 2^32
prod_prim = m_reduce(a_prim * b_prim) # divide a' * b' by 2^32
prod = m_reduce(prod_prim) # divide prod' by 2^32
print('%d * %d %% %d = %d' % (a, b, P, prod)) # prints 123456789 * 35 %
1000000007 = 320987587
```

Final remarks

One important issue that I've so far swept under the rug is that the output of `m_reduce` is actually in $[0, 2P)$ and not $[0, P)$. I just want end by discussing this issue. I can see two ways of handling this:

- Alternative 1. You can force $\text{m\_reduce}(a \cdot b)$ to be in $[0, P)$ for a and b in $[0, P)$ by adding an if-stament to the output of `m_reduce`. This will work for any odd integer $P < 2^{31}$.

  Fixed implementation of m_reduce
- Alternative 2. Assuming P is an odd integer $< 2^{30}$ then if a and $b \in [0, 2P)$ you can show that the output of $\text{m\_reduce}(a \cdot b)$ is also in $[0, 2P)$. So if you are fine working with $[0, 2P)$ everywhere then you don't need any if-statements. **Nyaan**'s github has a nice C++ implementation of Montgomery multiplication using this style of implementation.

montgomery, modular, fast multiplication, primes, constant factor