Modular Exponentiation

forthright48 on August 17, 2015

Problem

Given three positive integers B, P and M, find the value of $B^P \mod M$.

For example, $B = 2$, $P = 5$ and $M = 7$, then $B^P \mod M = 2^5 \mod 7 = 32 \mod 7 = 4$.

This problem is known as [ref1] Modular Exponentiation. In order to solve this problem, we need to have basic knowledge about [ref2] Modular Arithmetic.

Brute Force – Linear Solution $O(P)$

As usual, we first start off with a naive solution. We can calculate the value of $B^P \mod M$ in $O(P)$ time by running a loop from 1 to P and multiplying B to result.

```
1  int bigmodLinear ( int b, int p, int m ) {
2      int res = 1 % m;
3      b = b % m;
4      for ( int i = 1; i <= p; i++ ) {
5          res = ( res * b ) % m;
6      }
7      return res;
8  }
```

A simple solution which will work as long as the value of P is small enough. But for large values of P, for example, $P > 10^9$, this code will time out. Also note that in line 2, I wrote $res = 1 \mod m$. Some people tend to write $res = 1$. This will produce a wrong result when the value of P is 0 and value of M is 1.

Divide and Conquer Approach - $O(log_2(P))$

According to [ref3] Wiki,

> **A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type (divide), until these become simple enough to be solved directly (conquer).**

That is, instead of solving the big problem $B^P \mod M$, we will solve smaller problems of similar kind and merge them to get the answer to the big problem.

So how do we apply this D&C (Divide and Conquer) idea?

Suppose we are trying to find the value of B^P. Then three thing can happen.

1. **Value of P is 0 (Base Case)**: B^0 is 1. This will be the base case of our recursion function. Once we reach this state, we no longer divide the problem into smaller parts.
2. **Value of P is Even**: Since P is even, we can say that $B^P = B^{\frac{P}{2}} \times B^{\frac{P}{2}}$. For example, $2^{32} = 2^{16} \times 2^{16}$, $3^6 = 3^3 \times 3^3$. Therefore, instead of calculating the value of $x = B^P$, if we find the value of $y = B^{\frac{P}{2}}$, then we can get the value of x as $x = y \times y$.
3. **Value of P is Odd**: In this case we can simply say that $B^P = B \times B^{P-1}$.

Using these three states, we are can formulate a D&C code.

```
1  int bigmodRecur ( int b, int p, int m ) {
2      if ( p == 0 ) return 1%m; // Base Case
3  
4      if ( p % 2 == 0 ) { // p is even
5          int y = bigmodRecur ( b, p / 2, m );
6          return ( y * y ) % m; // b^p = y * y
7      }
8      else {
9          // b^p = b * b^(p-1)
10         return ( b * bigmodRecur ( b, p - 1, m ) ) % m;
11     }
12 }
```

In line 2, we have the base case. We return $1 \mod m$ in case value of m is 1. Next on line 4 we check if p is even or not. If it is even, we calculate $A^{\frac{P}{2}}$ and return after squaring it. In line 8, we handle the case when P is odd.

At each step we either divide P by 2 or subtract 1 from P and then divide it by 2. Therefore, there can be at most $2 \times log_2(P)$ steps in D&C approach. Hence complexity is $O(log_2(P))$.

A Better Solution

A better solution to this problem exists. By using the Repeated Squaring algorithm, we can find the Modular Exponentiation faster than D&C. Repeated Squaring Algorithm has the same complexity as D&C, but it is iterative, thus does not suffer from recursion overhead.

We need to know about bitwise operations before we learn Repeated Squaring algorithm. Since we did not cover this topic yet, I won't write about this approach now. Hopefully, once we cover bitwise operations I will write another post

Resource

1. Wikipedia - Modular Exponentiation
2. forthright48 - Introduction to Modular Arithmetic
3. Wikipedia - Divide and conquer algorithms
4. forthright48 - Introduction to Number Systems

Category: CPPS, Number Theory

Previous:
Leading Digits of Factorial

Next:
Contest Analysis: IUT 6th National ICT Fest 2014, UVa 12830 – 12839

Archives

May 2019 (1)
April 2019 (1)
March 2019 (1)
December 2018 (2)
November 2018 (4)
September 2018 (2)
February 2018 (1)
January 2018 (1)
November 2017 (2)
September 2015 (7)
August 2015 (13)
July 2015 (15)

Categories

CPPS (45)
- Combinatorics (4)
- Data Structure (1)
- Number Theory (36)

Meta (1)
Misc (4)

Recent Comments

My Shopee Interview – Shadman Protik on My Interview Experience with Shopee / Garena / Sea Group
Istiad Hossain Akib on SPOJ LCMSUM – LCM Sum
Rifat Chowdhury on MyStory#02 – Deciding Where to Study CS
Salman Farsi on Leading Digits of Factorial
Learning Notes on Multiplicative Functions, Dirichlet Convolution, Mobius Inversion, etc – RobeZH's Blog on SPOJ LCMSUM – LCM Sum