CS 681: Computational Number Theory and Algebra **Lecture 4: Chinese Remainder Theorem**
Lecturer: Manindra Agrawal **Notes by: Rohit Garg**
January 1, 2004.

1 Chinese Remainder Theorem

In today's lecture we will be talking about a new tool: Chinese Remaindering which is extremely useful in designing new algorithms and speeding up existing algorithms. Although Chinese Remainder Theorem is more known in reference with the integers, but the general statement of the theorem is as follows:

Let **R** be either a ring of integers(denoted by Z) or field of polynomials over field F (denoted by F[x]). Let m $\epsilon\,\mathbf{R}$ and m_1 , m_2 , . . . , m_k ϵ **R** be such that $(m_i\,,\ m_j) = 1$ for $i\ \neq\ j$ and m = $\prod_{i=1}^{k} m_i$.

Theorem 1.1 $\mathbf{R}/(m) \cong \mathbf{R}/(m_1) \oplus \mathbf{R}/(m_2) \oplus ..... \oplus \mathbf{R}/(m_k)$. *In words, this can be stated as the quotient ring of* **R** *mod principal ideal (m) is isomorphic to direct sum of quotient rings of* $\mathbf{R}/(m_1)$, $\mathbf{R}/(m_2)$, ... , $\mathbf{R}/(m_k)$.

Proof: Let $f_i : \mathbf{R}/(m) \longrightarrow \mathbf{R}/(m_i)$ be a mapping from the quotient ring of **R** mod ideal (m) to the quotient ring **R** mod ideal (m_i) as f_i(a + (m)) = a_i + (m_i) where a = a_i .
We can easily show that f_i to be a ring homomorphism.
We now define a mapping from the quotient ring of **R** mod ideal (m) to the direct sum of the quotient rings of **R** mod (m_1), . . , **R** mod (m_k) as follows
$f : \mathbf{R}/(m) \longrightarrow \mathbf{R}/(m_1) \oplus \mathbf{R}/(m_2) \oplus \ldots. \ \oplus \mathbf{R}/(m_k)$
$f(a + (m)) = (\ f_1(a + (m))\ ,\ f_2(a + (m))\ , \ldots,\ f_k(a + (m)))$

We claim that f is a ring isomorphism.
Since all the $f_i's$ are homomorphisms, therefore we have that f is also a homomorphism. We just need to show that f is 1-1 and onto.
Suppose f(a + (m)) = f(b + (m)).
Then f(a - b + (m)) = 0 which would further imply $f_i((a\ -\ b)\ +\ (m))\ =\ 0$ for all i.
$\Longrightarrow (a\ -\ b)\ \epsilon\ (m)$.
$\Longrightarrow f\ is\ 1 - 1$.
Now to show onto, given any element $(a_1\,,\,a_2\,,\,..,\,a_k)\,\epsilon\,\mathbf{R}/(m_1) \oplus \mathbf{R}/(m_2) \oplus \ldots. \oplus \mathbf{R}/(m_k)$, we will construct an element of **R** that would map to the above element.
Since $g.c.d(m_i\,,\,m_j)\ =\ 1$ for all j $\neq$ i and m = $\prod_{i=1}^{k} m_i$, g.c.d($\frac{m}{m_i}$, m_i) = 1. This implies that there exists an inverse of $\frac{m}{m_i}$ in $\mathbf{R}/(m_i)$.
Let the inverse be n_i such that $n_i\ *\ \frac{m}{m_i} = 1$ in $\mathbf{R}/(m_i)$.

We now define the pre-image of $(a_1\,,\,a_2\,,\,..\,,\,a_k)$ to be a = $\sum_{i=1}^{k} a_i \;*\; n_i \;*\; \frac{m}{m_i}$
It can be easily seen that f(a) = $(a_1\,,\,a_2\,,\,..\,,\,a_k)$ as f_i(a + (m)) = r
$where\,,\; r \;=\; a\, mod\,(m_i)$
$or\,,\; r \;=\; (\sum_{i=1}^{k} a_i \;*\; n_i \;*\; \frac{m}{m_i})\, mod\,(m_i)$
$or\,,\; r \;=\; \sum_{i=1}^{k}(a_i \;*\; n_i \;*\; \frac{m}{m_i}\, mod\,(m_i))$
$or\,,\; r \;=\; a_i \;*\; n_i \;*\; \frac{m}{m_i}\, mod\,(m_i)$
$or\,,\; r \;=\; a_i\, mod\,(m_i)\;\; (n_i \;*\; \frac{m}{m_i} \equiv\; 1\, mod\, m_i)$

■

Chinese Remainder Theorem has many uses:
1. It is used to divide big rings to many smaller rings which helps us to argue certain things in a much better manner as smaller rings can be handled well than larger rings.
2. It is also used to design new algorithms and many a times speed up existing algorithms.

To illustrate the use of Chinese Remainder Theorem, let us consider the problem of Matrix Multiplication.

Consider 2 matrices A and B with order $n \;\times\; n$ each.
Using a naive approach for multiplying 2 matrices would yield us an algorithm with a complexity of $O(n^3)$ operations. Moreover, assuming the entries of the matrices to be large(b bits long), the time complexity of the simple algorithm would then become $O(n^3 b^2)$.

But, the Chinese Remainder Theorem helps us to reduce the complexity by quite a good deal.
The basic idea of the algorithm would be to do all the computations modulo some small numbers and then combine the computed numbers using the Chinese Remainder Theorem to get the final result. So at the beginning we would need the number that would factorize and yield the corresponding relatively prime "small numbers".
Let m be required number. "m" should be atleast as large as the largest number possible arising out of the matrix multiplication.
Since the largest number possible after matrix multiplication is $n \times 2^{2b}$, therefore we choose m to be such that $m \;>\; 2n * 2^{2b}$. Therefore for simplicity purposes we choose m to be the product of first k prime numbers $p_1\,,\,p_2\,,\,\ldots\,,\,p_k$ such that m = $\prod_{i=1}^{k} p_i \;>\; 2n * 2^{2b}$.

Algorithm:

1. Find $p_1\,,\,p_2\,,\,p_3\,,\,..\,,\,p_k$ such that $\prod_{i=1}^{k} \;>\; 2n * 2^{2b}$
2. Compute A $*$ B (mod p_i) for each i = 1 to k.
3. Reconstruct A $*$ B using Chinese Remainder Theorem.

Time Complexity Analysis:
Step 1. From the *prime number theorem* (which states that the product of the primes less than a number t is asymtotically e^t), we can say that we would need to examine the primes only upto $\log(2n * 2^{2b})$. Using sieve of eratosthenes method this can be done in $O((b + \log(n))^2)$ time.

Step 2. In this step, we would first reduce the matrices and then multiply them.
Time for reducing each entry(b bits) in matrix by a prime ($\log b + \log\log n$ bits) = b($\log b + \log\log n$) time.
Since there are n^2 entries and k primes, therefore time for reducing the matrices = $O(kn^2b(\log b + \log\log n))$.

The resulting entries would now be of ($\log b + \log\log n$) bits.
Now, time to get a single entry of the resulting matrix afer multiplication is $(\log b + \log\log n)^2$. Since there are k primes and n^2 entries, the total time to obtain the matrix $A * B(mod p_i)$ for all i's is $O(kn^3(\log b + \log\log n)^2)$.

So, the total time for step 2 would be $O(\text{kb}(\log b + \log\log n)n^2) + O(kn^3(\log b + \log\log n)^2)$.

Step 3. In this step, we would need to compute m (which is the product of the primes), compute the inverses n_i of all $\frac{m}{m_i}$ and then apply the formula above mentioned(in the proof of the theorem) to get the pre-image.
Computing m would be same as multiplying a b bit number with a $\log b$ bit number and hence the time to get m would be $O(b \log b)$.
Computing each n_i would be of the order of $(\log b)^2$ and hence time to compute all the $n_i's$ would be $k(\log b)^2$
Now, there are n^2 entries and applying the formula to each entry would take O(kb) ignoring the log factors. Therefore, the total time to get the resulting matrix would be $O(kbn^2)$ which would be $O(n^2b^2)$.

So, total time complexity of the entire algorithm = $O(n^2b^2 + n^3b)$ which is far better than the complexity of the simple algorithm of multiplication having the complexity $O(n^3b^2)$.

The above example indicates that this trick of doing all the computations modulo small nos and then combining them using the Chinese Remainder Theorem helps us in reducing the time complexity in any computation involving large numbers. Further, this method yields a natural parallelization of the multiplication process.