Introduction to Discrete Structures
CSCI 3710

Spring Semester 2024 — CRN 20779

Honors Project

Goal

Create a software system that can perform basic arithmetic using residue number systems and measure performance of the system

Background

A *residue number system* uses modular arithmetic to reduce arithmetic operations on large integers into several operations on smaller integers.

Given a set of relatively prime moduli $m_1, m_2, \ldots, m_n$ and their product

$$M = \prod_{i=1}^{n} m_i$$

any integer $0 \leq x < M$ can be represented as a unique tuple of remainders mod m_i. Moreover, given $0 \leq x, y < M$, $x + y$, $x - y$ and $x \cdot y$ can be computed using remainders mod m_i. In other words:

- $x + y \equiv \langle a_1, a_2, \ldots, a_n \rangle$ where $(x + y) \equiv a_i \pmod{m_i}$
- $x - y \equiv \langle b_1, b_2, \ldots, b_n \rangle$ where $(x - y) \equiv b_i \pmod{m_i}$
- $x \cdot y \equiv \langle c_1, c_2, \ldots, c_n \rangle$ where $(x \cdot y) \equiv c_i \pmod{m_i}$

Note that the sum, difference and product have a unique representation modulo M. If the results of these operations are nonnegative and less than M, then the Chinese Remainder Theorem can be used to construct the result from the residues.

Details

One very important detail: for this project, all modulus values must be odd integers. No even numbers are allowed to be moduli. The reason for this will be explained later.

The project involves creating two classes — an *RNS* class to encapsulate the residue system and an *RNSNumber* class to represent one number within the given RNS.

Unless otherwise specified, all integers should be unsigned 32-bit integers. Notable exceptions are in performing modular arithmetic on residues, where unsigned 64-bit integers should be used, and in computing inverses, where signed 64-bit integers are necessary.

▹*The* RNS *class*

The RNS class should contain the following data:

- The list and count of moduli; these should be supplied to the constructor

- The list of $\hat{m}_i$ values, where $\hat{m}_i = (M/m_i) \bmod m_i$
- The list of inverse values $\hat{m}_i^{-1}$, where $\hat{m}_i \hat{m}_i^{-1} \equiv 1 \pmod{m_i}$
- The list of inverses of r_j, $2 \le j \le n$ where $m_1 r_j \equiv 1 \pmod{m_j}$
- The count of "vector" operations — the number of times `RNSNumber` objects are added, subtracted or multiplied
- The count of all arithmetic operations performed on residues; each operator on `RNSNumber` objects with n moduli should count as n operations in this category

In addition, the `RNS` class should support the following actions:

- A constructor that supplies the list of moduli and the number of moduli, either as an explicit parameter or via length of list
- A destructor that deletes all dynamically allocated memory
- Create an `RNSNumber` object whose value is given by an integer
- Create an `RNSNumber` object whose value is given by a string
- Take two `RNSNumber` objects and add, subtract or multiply them
- Get vector and scalar operation counts
- Take an `RNSNumber` object and convert it to string form
  - Note: I will provide code for this operation
- Take an `RNSNumber` object and compute its α value (see below)

▹*The* `RNSNumber` *class*

This class is basically just a container to hold the remainders of a number. Consider also including a reference to the `RNS` object that created the number object.

▹*Calculating* α

According to the Chinese Remainder Theorem, an integer x can be constructed using the following formula:

$$x \equiv \left| \sum_{i=1}^{n} \left| a_i \hat{m}_i \hat{m}_i^{-1} \right|_{m_i} \right|_M$$

where $|x|_m$ represents $x \bmod m$. All answers are then given by

$$x = \sum_{i=1}^{n} \left| a_i \hat{m}_i \hat{m}_i^{-1} \right|_{m_i} + kM$$

for some integer k.

Let α be the unique integer such that

$$0 \le x = \sum_{i=1}^{n} \left| a_i \hat{m}_i \hat{m}_i^{-1} \right|_{m_i} - \alpha M < M$$

Rearranging terms gives

$$\begin{aligned}
x &= \sum_{i=1}^{n} \left| a_i \hat{m}_i \hat{m}_i^{-1} \right|_{m_i} - \alpha M \\
\alpha M &= \sum_{i=1}^{n} \left| a_i \hat{m}_i \hat{m}_i^{-1} \right|_{m_i} - x \\
\alpha &= \sum_{i=1}^{n} \frac{\left| a_i \hat{m}_i \hat{m}_i^{-1} \right|_{m_i}}{M} - \frac{x}{M} \\
&= \sum_{i=1}^{n} \frac{\left| a_i \hat{m}_i^{-1} \right|_{m_i}}{m_i} - \frac{x}{M} \\
&= \left\lfloor \sum_{i=1}^{n} \frac{\left| a_i \hat{m}_i^{-1} \right|_{m_i}}{m_i} \right\rfloor
\end{aligned}$$

The last line follows because α is an integer and $x/M < 1$.

In a perfect world this gives us the desired answer. However, we run into a problem with rounding error; moreover, this is an issue regardless of whether we use integers or floating-point arithmetic.

"Current" solutions (scare quotes because that goes back 30+ years) all use floating-point arithmetic and use various techniques for handling the resulting error. Our approach will use only integer arithmetic.

Note that as written the result will always be zero since the numerator of each term is less than m_i. To compensate, multiply each numerator by 2^{32} and divide the resulting sum by 2^{32}, preserving any fractional component in a separate integer.

Each integer division introduces truncation error; although we cannot know the exact amount of each individual division's error, we do know that each is not more than 2^{-32}, the inverse of the scale factor. We also know that there are n divisions since there are n moduli involved. Thus, the worst case error is $n \cdot 2^{-32}$.

Now, consider the scaled summation:

$$\alpha' + \frac{f}{2^{32}} = \frac{1}{2^{32}} \sum_{i=1}^{n} \left\lfloor \frac{2^{32} \left| a_i \hat{m}_i^{-1} \right|_{m_i}}{m_i} \right\rfloor$$

where α' is the integer portion of the result and $0 \le f < 2^{32}$ is the numerator of the fractional part of the result. Then, $\alpha = \lfloor \alpha' + f/2^{32} + \varepsilon \rfloor$, where ε is the total truncation error.

If $f < 2^{32} - n$, then $f/2^{32} + \varepsilon < 1$ regardless of the actual error magnitude; in this case, $\alpha = \alpha'$ and we are done.

▸*What if $f \ge 2^{32} - n$?*

TBA; for now, focus on creating the classes. More info to follow soon.

What To Turn In

TBA, but definitely the code in a compressed folder.