System of simultaneous congruences

- Find x that solves following system?
- $$\begin{cases} x \equiv 2 \pmod{3} \\ x \equiv 3 \pmod{4} \\ x \equiv 1 \pmod{5} \end{cases}$$ modes pairwise gcd = 1, gcd(3,4) = 1,gcd(4,5) = 1, gcd(3,5) = 1
- For each one, find all solutions and intersect!
- x ∈{2, 5, 8, 11, 14, …..71 …} from first
- x ∈{3, 7, 11, 15, 19, 23, 27, 3…..71..
- x ∈{1, 6, 11, 16, 21, 26, 31, 36……71...
- Intersect: x ∈{11, 71, ...} => x ≡ 11 (% 60)

System of simultaneous congruences

```
bool satisfySystem(int x, vector<int> &rems, vector<int> &mods) {
    for (int i = 0; i < (int)mods.size(); ++i) {
        if(x % mods[i] != rems[i])  // x = rem[i] (% mods[i])
            return false;
    }
    return true;
}

// Find x satisfies
// x = rem[0] (% mods[0])
// x = rem[1] (% mods[1])
// ....
// x = rem[n] (% mods[n])
int solveSystemOfCongruences(vector<int> &rems, vector<int> &mods) {
    for(int x = 0 ; ; ++x) {
        if(satisfySystem(x, rems, mods))
            return x;
    }
    return -1;  // will never happens under some conditions
}
```

Chinese remainder theorem

- For a System of simultaneous congruences of The theorem till us solution exist in **2 cases**
- $$\begin{cases} x \equiv a_1 & \pmod{n_1} \\ \quad \cdots \\ x \equiv a_k & \pmod{n_k} \end{cases}$$
- 1) n1, ..., nk are positive integers that are pairwise coprime … or
- 2) $a_i \equiv a_j \pmod{\gcd(n_i, n_j)} \qquad \text{for all } i \text{ and } j$
- X are then congruent modulo the **LCM** of ni

Chinese remainder theorem

- System: $x \equiv A[i] \pmod{M[i]}$
  - $x \equiv 2 \pmod 3$ (assuming pairwise coprimes)
  - $x \equiv 3 \pmod 4$
  - $x \equiv 1 \pmod 5$
- Step 1: For the **ith** equation, compute **product** of ALL modes, except the current equation
  - X = 4*5 + 3*5 + 3*4 [e.g. 1st 3*4*5 / 3]
  - Then when we take a mode, 2 terms cancels and 1 remain
  - Divide the remaining term & Multiply needed reminder
  - Then we end with needed reminder per a term

Chinese remainder theorem

- X = 4*5 + 3*5 + 3*4 mods [3, 4, 5]
  - (4*5 + 0 + 0) % 3 [other 2 terms has 3, so = 0]
  - (0 + 3*5 + 0) % 4
  - (0 + 0 + 3*4) % 5
- X = (20, 15, 12) vs (2, 3, 1)
- Convert to (20/20 * 2, 15/15 * 3, 12/12 * 1)
- 1/20 % 3 = **2** - 1/15 % 4 = **3** - 1/12 % 5 = **3**
- X = 20 * **2** * 2 + 15 * **3** * 3 + 12 * **3** * 1 = **251**
- Intuition: min X divisible by 3, 4, 5? lcm (3, 4, 5)
- From theorem: 251 % 60 = 11 (the min X)

Chinese remainder theorem

```
// Given set of relative primes mod, solve the system of congruence using CRT
ll solveSystemOfCongruences_ch1(vector<ll> &rems, vector<ll> &mods) {
    ll prod = 1, x = 0;

    for(auto mod : mods)
        prod *= mod;

    for (int i = 0; i < (int)mods.size(); i++) {
        ll subProd = prod / mods[i];
        x += subProd * modInversek(subProd, mods[i]) * rems[i];
    }

    return x % prod;
}
```

Chinese remainder theorem

- Previous method handles only when modes are co-prime, but not the general **restricted** form $a_i \equiv a_j \pmod{\gcd(n_i, n_j)} \quad \text{for all } i \text{ and } j$
- If we can solve **2 Congruence** equation and **merge** in 1, we can solve sequentially
  - T = x mod N => T=N*k+x
  - T = y mod M => T=M*p+y
  - N*k+x = M*p+y => N*k-M*p=y-x LDE
  - New mod: use LCM of (N, M). New rem: (T=N*k+x)%M
- Once merged, move to next equation

Chinese remainder theorem

```
// If we can solve 2 cong equation and merge in 1, we can solve sequentially
// T = x mod N      => T=N*k+x
// T = y mod M      => T=M*p+y
// N*k+x = M*p+y    => N*k-M*p=y-x    => Linear Diophantine equation
ll solveSystemOfCongruences_NOT_RELATIVES(vector<ll> &rems, vector<ll> &mods) {
    ll rem = rems[0], mod = mods[0];

    // solve with prev equ, get new congruence equ
    for (int i = 1; i < (int)rems.size(); i++) {
        ll x, y, found, a = mod, b = -mods[i], c = rems[i] - rem;
        ll g = ldioph(a, b, c, x, y, found);

        if(!found)
            return -1;

        rem += mod * x;             // Evaluate previous congruence
        mod = mod / g * mods[i];    // merged mod: lcm modes so far
        rem = (rem%mod+mod)%mod;    // merged rem
    }
    return rem;
}
```

Chinese remainder theorem

- There are other ways to handle CRT
- Solve sequentially, and for each 2 equations, get GCD of modes, and generate 4 equations and solve…..let D = GCD(N, M)
  - $T \equiv (x \% D) \bmod D$
  - $T \equiv (x \% (N / D)) \bmod (N / D)$
  - $T \equiv (y \% D) \bmod D$
  - $T \equiv (y \% (M / D)) \bmod (M / D)$
- Variant of substitution method
- Garner Algorithm (Fast - coprimes only ?)

CRT usage

- Compute F()%C where C is not prime?
  - Assume we can solve F() % p^a
  - Factorize C: e.g. C = 12 = 2*2*3
  - Divide to co-primes list: e.g. {4, 3}
  - Now compute F() % 4 and F() % 3
  - But we need F()%12? CRT can solve this system
- See example, fermat and euler

CRT usage

- Assume we solve F(), its result that fit in 32 bit. But you notice intermediate overflow
- Pick M = P1*p2...Pk….set of prime numbers
  - such that M needs > 32 bit integer
  - The F() % M = F()
  - E.g. M = 257 * 263 * 269 * 271
- Compute F()%Pi: hence no overflow
- Use CRT to get the actual F()