Number theory

- Format
  - This lecture - now
  - Practice problems - available now (see description)
  - Stream - in ~2.5 days
- General tip: **model it as an equation** (if possible)

Floor/ceil

$\lfloor x \rfloor = x$, rounded down (floor)

$\lfloor \frac{5}{2} \rfloor = \lfloor 2.5 \rfloor = 2$, $\lfloor \frac{4}{2} \rfloor = \lfloor 2 \rfloor = 2$

$\lceil x \rceil = x$, rounded up (ceil)

$\lceil \frac{5}{2} \rceil = \lceil 2.5 \rceil = 3$, $\lceil \frac{4}{2} \rceil = \lceil 2 \rceil = 2$

Divisors (basic definitions)

- a | b <-> “a divides b” <-> “b is divisible by a” <-> b % a = 0
- Divisor (or factor) of x: a number y where y | x
- Transitive: a | b, b | c -> a | c
- Prime: no divisors except 1 and itself
- Prime factorization: breaking a number down into a product of primes (or prime powers: p^k for prime p and integer k)
    - ex: 2 = 2, 10 = 2 * 5, 254436 = 2^2 * 3 * 7 * 13 * 233
    - 27 = 3 * 9 **not** valid, 9 can be further broken down, should be 27 = 3^3

Prime factorization

- Goal: break x down into its prime divisors
- Naive - try all numbers up to x
- A bit less naive - try only the primes up to x
- Even less naive - you can't have two primes larger than sqrt(x). So you only have to check up to sqrt(x), then you can conclude that x is prime if it has no other prime factors.
- Complexity: O(sqrt(x) / log(x)) [note, O(log(sqrt(x))) = O(log(x))]
- Expected to have O(x / log(x)) primes within [1, x]

Prime factorization - code

```
vector<ll> factors(ll x) {
    vector<ll> ret;
    for (ll p: primes) { // precompute list of primes
        if (p * p > x) {
            // checked all primes below p, now x can't be a product of two unchecked primes since p^2 > x
            // so x must be prime

            if (x > 1) {
                ret.push_back(x);
                x = 1;
            }
        } else {
            while (x % p == 0) { // take as much of this prime as we have
                ret.push_back(p);
                x /= p; // make sure to divide x, that makes the p^2 > x case work
            }
        }
    }
    return ret;
}
```

More prime factorization

- More algorithms, not covered here, with better complexities
- See description for some of those algorithms
- Also, see description for some wicked fast factorization code

Finding divisors

- Similar to prime factors, but find all divisors of x
- If you know y is a divisor, then x / y is too
- Only have to check up to sqrt(x), the rest will be found as x / y
- Complexity: O(sqrt(x))
- Another algorithm: prime factorize x, then generate the divisors from that (DFS or something)
- If implemented well, O(number of divisors)

Modulo

- % - modulo
- x % m is the remainder of division of x / m
- ex: 17 / 6 = 2 with remainder 5, so 17 % 6 = 5
- Math requires: 0 <= (x % m) < m
- 17 % 6 = (17 - 6 - 6) = 5, -15 % 6 = (-15 + 6 + 6 + 6) = 3
- Another representation: x = am + b [a, b integers; 0 <= b < m]
- 17 = 2 * 6 + 5, -15 = -3 * 6 + 3
- Since [am] is a multiple of m, we just drop it when modding
- Since 0 <= b < m, it doesn’t change when modding
- So (am + b) % m = b
- In C++, -15 % 6 gives you -3, but it should be 3… be careful

Modulo - practice

- **Task**: using the x = am + b representation, prove:
  - (a + b) % m = ((a % m) + (b % m)) % m
  - (a - b) % m = ((a % m) - (b % m)) % m
  - (a * b) % m = ((a % m) * (b % m)) % m
  - What about division? We’ll get to that soon...

Modulo - practice

- $(a * b) \% m = ((a \% m) * (b \% m)) \% m$
  - Let $a = cm + d$, $b = em + f$
  - $(a * b) \% m$
    - $((cm + d) * (em + f)) \% m$
    - $(cem^2 + cfm + dem + df) \% m$
    - Everything but df is a multiple of m, so we just get $df \% m$
  - $((a \% m) * (b \% m)) \% m$
    - $(((cm + d) \% m) * ((em + f) \% m)) \% m$
    - $(((d) \% m) * ((f) \% m)) \% m$
    - $0 <= d < m$, $0 <= f < m$, so $d \% m = d$, $f \% m = f$
    - And we’re left with $df \% m$ again
  - (so, left side = right side)

More modulo

- Binary exponentiation - compute a^b % m quickly
  - Write b in binary, ex: $21_{10} = 10101_2$
  - So $a^{21} = a^1 * a^4 * a^{16}$
  - $a^{2x} = a^x * a^x$, so we can compute a^1, a^2, a^4, a^8, etc. in O(1) each
  - Take modulo with each multiplication
  - Total complexity: O(log b)
- What about something like a^(b^c) % m? b^c has to be computed modulo ϕ(m) - totient function, see description
- Logarithm? Complicated, not covered here...

Modular division?

- weird
- Inverse of multiplication - modular multiplicative inverse
- $(1 / x) \% m = x^{-1} \% m =$ the y such that $xy \% m = 1$
- Normally, $x * (1 / x) = 1$, so this works
- $2^{-1} \% 4$ doesn't exist? When does it?
- Only exists if $\gcd(x, m) = 1$ (we'll get to gcd right after this)
  - Otherwise, any xy will be divisible by $\gcd(x, m)$ and can't be 1

Modular division - computation

- Want: y such that xy % m = 1
- For **prime** m
  - Fermat's little theorem: a^{m-1} % m = 1 for any a, 0 < a < m (proof in description)
  - So divide both sides by a: $a^{m-2} = a^{-1}$ (mod m)
  - That's it, it's just $y = x^{m-2}$
- For **coprime** x, m (gcd(x, m) = 1)
  - Extended euclidean algorithm, covered soon in this video

GCD

- “Greatest common divisor”: gcd(x, y) = largest g where g | x, g | y
- If $x = p_1^a p_2^b \ldots$ and $y = p_1^c p_2^d \ldots$, then $\gcd(x, y) = p_1^{\min(a, c)} p_2^{\min(b, d)} \ldots$
- Uses the Euclidean algorithm
- gcd(a, b, c, …) = gcd(gcd(a, b), c, …)

Euclidean GCD

- gcd(a, b) = gcd(a - b, b)
  - Let the GCD be g, then g | a and g | b, so a % g = 0, b % g = 0
  - (a - b) % g = ((a % g) - (b % g)) % g = 0, so g | (a - b)
  - Some casework to show g doesn’t increase
- Mod is repeated subtraction, so… gcd(a, b) = gcd(a % b, b)
- Generally assumed a >= b, so we use gcd(a, b) = gcd(b, a % b)
- gcd(x, 0) = x, base case
- If a >= b, a % b <= a / 2, halves each time
- O(log(min(a, b)))
- Two numbers x, y are **coprime/relatively prime** if gcd(x, y) = 1

Euclidean GCD - example

- gcd(133, 56) [133 = 7 * 19, 56 = 2^3 * 7]
- gcd(133, 56) = gcd(56, 133 % 56) = gcd(56, 21)
- gcd(56, 21) = gcd(21, 56 % 21) = gcd(21, 14)
- gcd(21, 14) = gcd(14, 21 % 14) = gcd(14, 7)
- gcd(14, 7) = gcd(7, 14 % 7) = gcd(7, 0)
- gcd(7, 0) = 7, so gcd(133, 56) = 7

```
ll gcd(ll a, ll b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

Extended Euclidean algorithm

- Solves: ax + by = 1 [a, b given; x, y unknown; gcd(a, b) = 1]
- More generally: ax + by = gcd(a, b)
- Bezout's identity: says this solution (x, y) will always exist
- More info in description, but basically run GCD backwards
- So, want to solve for z: cz % m = 1
- Run extended Euclidean with a = c, b = m, solving cx + my = 1
- [my] factor lets you add and subtract m freely, just like mod
- Alternatively, take both sides mod m, you get exactly cx % m = 1
- Then, z = x
- Also O(log(min(a, b)), same idea as GCD

LCM

- "Least common multiple": lcm(x, y) = smallest z where $x \mid z$, $y \mid z$
- If $x = p_1^a p_2^b \ldots$ and $y = p_1^c p_2^d \ldots$, then $\mathrm{lcm}(x, y) = p_1^{\max(a, c)} p_2^{\max(b, d)} \ldots$
- lcm(a, b) = a * b / gcd(a, b)
- lcm(a, b, c, …) = lcm(lcm(a, b), c, …)

Chinese remainder theorem

- Given a, b, x, y [0 <= a < x; 0 <= b < y; x, y coprime]
- Theorem states that there exists unique z where:
  - z % x = a
  - z % y = b
- Not restricted to just 2 equations
- If not coprime, can be invalid, impossible for z % 4 = 1, z % 6 = 0
- How to find z?
- **Task**: try to derive a way to find z, expected complexity better than O(x) or O(y)

Chinese remainder theorem - derivation

Hint 1: Remember the modulo form, $x = am + b$ for $b = x \% m$

Chinese remainder theorem - derivation

Hint 2: Reduce it to some algorithm (covered in this video) that runs in $O(\log(\min(x, y)))$

Chinese remainder theorem - derivation

- Look at the case with only 2 equations - if there are more, do 2 at a time
- Recall: task is to find z where
  - z % x = a
  - z % y = b
- Let’s satisfy the first equation, so z = cx + a (for some integer c)
- Plug that into second: (cx + a) % y = b
- Subtract a: cx % y = (b - a) % y
- We can solve for dx % y = 1 (since gcd(x, y) = 1), so let’s find d
- Now, set c = (b - a) * d, which is thus equal to (b - a) [mod y]
- Plug c into z = cx + a, and get z = d(b - a)x + a
- Complexity: O(log(min(x, y)))

Möbius inversion (kinda?)

- Ex: You have an array, length up to 10^5, array elements up to 10^5. Find the number of subsequences where the overall GCD of elements is 1. (problem link in description)
- A modification: let all a[i] <= m, where m <= 10^5. For each x from 1 to m, find the number of subsequences where the overall GCD of elements is x

Möbius inversion (kinda?)

- Ex: $n = 5$, $m = 6$, $a = [2, 5, 3, 6, 1]$
- $f(x)$ = answer for x
- $f(1) = 2^5 - 1$? but that counts stuff like {2, 6}
- This really gives us $g(x)$, which is $f(x)$ over all multiples of x
- So let’s get $f(x)$ by subtracting out the multiples from $g(x)$
    - $f(6) = g(6)$
    - $f(5) = g(5)$
    - $f(4) = g(4)$
    - $f(3) = g(3) - f(6)$
    - $f(2) = g(2) - f(4) - f(6)$
    - $f(1) = g(1) - f(2) - f(3) - f(4) - f(5) - f(6)$
- And that’s it.

Subtracting multiples?

- Can we just subtract out the multiples in less than O(n^2)?
- Subtracting out multiples of i is O(m / i)
- As it turns out,

$$\sum_{i=1}^{m} \frac{m}{i} = O(m \log(m))$$

- (proof in description)
- We also need to count the number of multiples of i for each i, but this can also be done in O(m / i) using a frequency array
- So that’s the complexity - O(m log(m))

Conclusion

- Do problems
- Stream soon
- Bye

Number Theory

$$\gcd(133, 56)$$
$$= \gcd(56, 21)$$
$$= \gcd(21, 14)$$
$$= \gcd(14, 7)$$
$$= \gcd(7, 0)$$
$$= 7$$

$$x = p_1^{e_1} p_2^{e_2} p_3^{e_3} \cdots$$
$$254436 = 2^2 \cdot 3 \cdot 7 \cdot 13 \cdot 233$$

$$z \% 5 = 4$$
$$z \% 7 = 6$$
$$\therefore z = 34$$