Number Theory

Farhan Ahmad, Nafis Ul Haque Shifat and Debojoti Das Soumya

21st January 2022

সূচীপত্র

1 Divisibility and Factorization

Debojoti Das Soumya

1.1 Divisor বা Factor বা উৎপাদক

একটি সংখ্যা অন্য একটি সংখ্যার উৎপাদক হবে যদি এই সংখ্যাটি অন্য সংখ্যাকে নিঃশেষে ভাগ করে। অর্থাৎ a, b এর উৎপাদক যদি $a|b$। এখানে | সাইন দ্বারে বুঝায় a divides b বা a, b কে নিঃশেষে ভাগ করে। একটি উদাহরণ দেয়া যাক, যেমন 36 এর উৎপাদক হবে $1, 2, 3, 4, 9, 12, 18, 36$। উক্ত সব সংখ্যাই 36 কে নিঃশেষে ভাগ করে, তাই এই সব সংখ্যাই 36 এর উৎপাদক। উৎপাদক তো বুঝা গেল, কিন্তু আমরা প্রোগ্রাম দিয়ে কিভাবে চেক করব যে একটি সংখ্যা অন্য সংখ্যার উৎপাদক? a যদি b এর উৎপাদক হয় তাহলে b কে a দ্বারা ভাগ করলে ভাগশেষ 0 হবে। তাই আমরা বলতে পারি $b \equiv 0 \pmod{a}$। প্রোগ্রামিং এর ভাষায় *b % a == 0* হলে b, a এর উৎপাদক।

1.2 Multiple বা গুণিতক

গুণিতক হচ্ছে উৎপাদক এর বিপরীত। একটি সংখ্যা যদি আরেক্টি সংখ্যাকে নিঃশেষে ভাগ করে তাহলে দ্বিতীয় সংখ্যাটি প্রথমটির গুণিতক। অর্থাৎ যদি $a|b$, তাহলে b, a এর গুণিতক। যেমন 5 এর গুণিতক হলো $5, 10, 15, 20, \ldots, 100, \ldots$। বুঝাই যাচ্ছে একটি সংখ্যার অসংখ্যটি গুণিতক থাকবে। একটি সংখ্যা a এর গুণিতক গুলো হবে $a, 2a, 3a, \ldots$ অর্থাৎ ka form এ। প্রোগ্রামিং এর ভাষায় একটি সংখ্যা অন্য সংখ্যার গুণিতক কিনা তা কিভাবে চেক করব এটা তোমরা নিজেরা ভেবে দেখ।

1.3 GCD (Greatest Common Divisor)

দুইটি সংখ্যা a এবং b এর GCD হচ্ছে সবচেয়ে বড় সংখ্যা c যেন $c|b$ এবং $c|a$। আমরা সুবিধার জন্য দুইটা সংখ্যার GCD কে $gcd(a, b)$ দিয়ে প্রকাশ করব। যেমন $gcd(30, 45) = 15$। GCD বের করার জন্য যে এলগরিদম ব্যবহার করা হয় তার নাম Euclidean Algorithm। এই এলগরিদম থেকে পাওয়া যায় যে, $gcd(a, b) = gcd(b - a, a)$ বা $gcd(a, b) = gcd(b \mod a, a)$ (এভাবে বিয়োগ করতে করতে একসময় আমরা ভাগশেষ পাব)। এভাবে করলে আমাদের প্রোগ্রামের complexity দাড়ায় $O(\log \max(a, b))$। নিচে কোডটি দেয়া হলোঃ

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

GCD বের করার জন্য C++ এ একটি efficient built-in function আছে এটি ব্যবহার করতে *__gcd(a, b)* call করো। এটিরও complexity $O(\log \max(a, b))$।

1.4 LCM (Lowest Common Multiple)

দুইটি সংখ্যা a এবং b এর LCM হচ্ছে সবচেয়ে ছোট সংখ্যা c যেন $a|c$ এবং $b|c$। আবারো আমরা সুবিধার জন্য দুইটি সংখ্যার LCM কে $lcm(a, b)$ দিয়ে প্রকাশ করব। যেমন $lcm(10, 25) = 50$। GCD ও LCM

নিয়ে একটি সূত্র আছে যে, $ab = gcd(a,b) \cdot lcm(a,b)$ বা $lcm(a,b) = \frac{ab}{gcd(a,b)}$ (factorization পড়ার পর প্রমাণটি নিজে চেষ্টা করতে পার)। এখন আমরা সহজেই LCM বের করতে পারি।

1.5 Finding Divisors

সমস্যাঃ আমাদেরকে একটি ধনাত্মক সংখ্যা n এর সব উৎপাদক বের করতে হবে।

যেহেতু একটি সংখ্যার সব উৎপাদক ওই সংখ্যা থেকে ছোট বা সমান। তাই, এই সমস্যার সবচেয়ে সহজ সমাধান হবে 1 থেকে n পর্যন্ত সব সংখ্যা চেক করা যে এটা কী n এর উৎপাদক কিনা। এভাবে করে আমরা $O(n)$ এই n এর সব উৎপাদক বের করে ফেলতে পারি। যদি $n \leq 10^9$ হয় তাহলে এই সমাধান অনেক slow। আমাদেরকে এই সমাধানকে optimize করতে হবে।

একটি জিনিস খেয়াল করি যদি, $x|n$ তাহলে $\frac{n}{x}|n$ কারণ $x \cdot \frac{n}{x} = n$। তাহলে প্রতি উৎপাদক a এর জন্য অন্য একটি সংখ্যা b থাকবে যেন $ab = n$ হয়। আরেক্টি বিষয় হচ্ছে a ও b এর মধ্যে কোন একটি সংখ্যা $\sqrt{n}$ থেকে ছোট বা সমান হবে কারণ যদি $a > \sqrt{n}$ এবং $b > \sqrt{n}$ হয় তাহলে $ab > \sqrt{n} \cdot \sqrt{n}$ বা $ab > n$। কিন্তু আমরা প্রথমেই বলেছি যে $ab = n$। তাহলে দুটি সংখ্যাই $\sqrt{n}$ থেকে বড় হওয়া সম্বব নয়।

তো আমরা এখন 1 থেকে $\sqrt{n}$ পর্যন্ত লুপ চালাবো যদি i, n এর উৎপাদক হয় তাহলে $\frac{n}{i}$ ও n এর উৎপাদক হবে (যদি $i = \frac{n}{i}$ হয় তাহলে $\frac{n}{i}$ নিয়ে আবার কাজ করার দরকার নাই কারণ i দিয়ে একবার কাজ করা হয়ে গেছে)। এভাবে আমরা $O(\sqrt{n})$ এ n এর সব উৎপাদক বের করে ফেলতে পারি। নিচে কোডটি দেয়া হলোঃ

```
for (int i = 1; i * i <= n; i++) { // i <= sqrt(n)  -> i * i <= n
  if (n % i == 0) {
    // i is a factor
    if (i != n / i) {
      // n / i is another factor
    }
  }
}
```

1.6 Prime numbers বা মৌলিক সংখ্যা

মৌলিক সংখ্যা হলো এমন সব ধনাত্মক সংখ্যা p যেন 1 ও p ছারা p এর আর কোন উৎপাদক না থাকে। $2, 3, 5, 7, 11 \ldots$ হলো প্রথম কয়টি মৌলিক সংখ্যা। একটি মৌলিক সংখ্যা p এর উৎপাদক সংখ্যা 2 (1 মৌলিক সংখ্যা নয়)। তাই আমরা আগের মতো উৎপাদক সংখ্যা গুনে বের করে ফেলতে পারব একটি সংখ্যা মৌলিক কিনা।

1.7 Divisors of all numbers from 1 to n

সমস্যাঃ আমাদেরকে একটি ধনাত্মক সংখ্যা n দেয়া হবে আমাদেরকে 1 থেকে n পর্যন্ত সব সংখ্যার উৎপাদক বের করতে হবে।

আগের মতো আমরা 1 থেকে n পর্যন্ত লুপ চালিয়ে সব সংখ্যার উৎপাদক বের করতে পারি। তাহলে আমাদের complexity দাড়ায় $O(n\sqrt{n})$। কিন্তু আমরা এটি $O(n \log n)$ এই বের করে ফেলতে পারি।

আমরা প্রতি সংখ্যা $m(1 \le m \le n)$ এর জন্য একটি *vector* রাখব যার মধ্যে m এর সব উৎপাদক থাকবে। 1 থেকে n পর্যন্ত লুপ চালাবো এবং দেখব i কাদের উৎপাদক এবং তাদের *vector* এ আমরা i কে insert করে দিব। এখন i কাদের কাদের উৎপাদক এটা কিভাবে বের করা যায়? খুবি সোজা i যদি x এর উৎপাদক হয় তাহলে x, i এর multiple আর একটি সংখ্যার সব multiple আমরা জানি যা হলো $i, 2i, 3i, \ldots$। এখানে সব সংখ্যার দরকার নাই শুধু $ki \le n$ গুলো দরকার। তাহলে আমাদের time complexity কতো?

$$\begin{aligned}\sum_{i=1}^{n} \frac{n}{i} &= \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n-1} + \frac{n}{n} \\ &= n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}\right) \approx n \log n\end{aligned}$$

$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}$ কে Harmonic Series বলা হয়। এই Series এর প্রথম n term এর যোগফল approximately $\log n$। কোডটি নিচে দেয়া হলোঃ

```
vector<int> divisors[n + 1]; // divisors[i] will store the divisors of i
for (int i = 1; i <= n; i++) {
  for (int j = i; j <= n; j += i) { // iterating over the multiples of i
    divisors[j].push_back(i); // i is a divisor of j so add i to the divisor list
    // of j
  }
}
// divisors[i] stores the divisors of i
```

1.8 Prime Factorization

1.8.1 Fundamental Theorem of Arithmetic

1 বাদ দিয়ে যেকোন ধনাত্মক সংখ্যার একটি ইউনিক prime factorization আছে ওই সংখ্যা কে নিচের form এ লিখা সম্ভব।

$$p_1^{a_1} p_2^{a_2} p_3^{a_3} \cdots p_k^{a_k} = \prod_{i=1}^{k} p_i^{a_i}$$

এখানে প্রতিটি p_i একটি মৌলিক সংখ্যা এবং $a_i > 0$। যেমন $180 = 2^2 3^2 5^1$।

1.8.2 Factorizing a number

একটি সংখ্যা n যদি মৌলিক না হয় তাহলে $\sqrt{n}$ এর সমান বা ছোট n এর একটি মৌলিক উৎপাদক থাকবে এবং একটি সংখ্যার সবচেয়ে ছোট উৎপাদক একটি মৌলিক সংখ্যা। এই property গুলো ব্যবহার করে আমরা একটি সংখ্যার prime factorization $O(\sqrt{n})$ এ বের করে ফেলতে পারি। নিচে কোডটি দেওয়া হলোঃ

```
vector<pair<int, int>> p; // for each prime divisor stores (prime, power)
for (int i = 2; i * i <= n; i++) {
  if (n % i == 0) { // i is a prime factor of n
    int power = 0;
    while (n % i == 0) { // calculates power
      n /= i;
      power++;
    }
    p.push_back({i, power});
  }
}
if (n > 1) p.push_back({n, 1}); // n is a prime factor
```

একটা উদাহরণের মাধ্যমে কোডটি বুঝা যাক।
$N = 2205 = 3^2 5^2 7$

$i = 2$: nothing happens
$i = 3$:
  $power = 2$
  $p = [(3, 2)]$
  $N = 245 = 5^2 7^1$
$i = 4$: nothing happens
$i = 5$:
  $power = 2$
  $p = [(3, 2), (5, 2)]$
  $N = 7^1$
$i = 6$:
  $i \cdot i > N$ loop stops

$N \neq 1$:
  $p = [(3, 2), (5, 2), (7, 1)]$

1.8.3 Divisors using prime factorization

যদি $n = p_1^{a_1} p_2^{a_2} p_3^{a_3} \cdots p_k^{a_k}$ হয় আর যদি $d|n$, তাহলে $d = p_1^{b_1} p_2^{b_2} p_3^{b_3} \cdots p_k^{b_k}$ হবে যেখানে $0 \leq b_i \leq a_i$।

1.8.4 LCM and GCD using prime factorization

যদি $n = \prod_{i=1}^{k} p_i^{a_i}$ এবং $m = \prod_{i=1}^{k} p_i^{b_i}$ হয় তাহলেঃ

$$lcm(n, m) = \prod_{i=1}^{k} p_i^{\max(a_i, b_i)}$$

$$gcd(n, m) = \prod_{i=1}^{k} p_i^{\min(a_i, b_i)}$$

1.9 Some number theoretic functions

1.9.1 Number of divisors

যদি $n = p_1^{a_1} p_2^{a_2} p_3^{a_3} \cdots p_k^{a_k}$ হয় তাহলে n এর উৎপাদক সংখ্যা হলোঃ

$$d(n) = (a_1 + 1)(a_2 + 1) \cdots (a_{k-1} + 1)(a_k + 1) = \prod_{i=1}^{k} (a_i + 1)$$

1.9.2 Sum of divisors

যদি $n = p_1^{a_1} p_2^{a_2} p_3^{a_3} \cdots p_k^{a_k}$ হয় তাহলে n এর উৎপাদক এর যোগফল হলোঃ

$$\sigma(n) = (1+p_1+p_1^2+\cdots+p_1^{a_1})(1+p_2+p_2^2+\cdots+p_2^{a_2}) \cdots (1+p_k+p_k^2+\cdots+p_k^{a_k}) = \prod_{i=1}^{k} (1+p_i+p_i^2+\cdots+p_i^{a_i})$$

$$= \prod_{i=1}^{k} \sum_{j=0}^{a_i} p_i^j$$

2 Sieve Of Eratosthenes

Farhan Ahmad

Sieve নামটা অনেকেই হইত আগেই শুনেছ। অনেক জনপ্রিয় একটা algorithm প্রাইম বের করার জন্য। যেইটা [1, N] range এর সব প্রাইম বের করে।

এখানে, আমরা প্রাইম এর যেই বৈশিষ্ট্যর অপর নির্ভর করব, টা হল যেকোনো প্রাইম এর মাত্র দুইটি divisor থাকে। তাই অন্য কোন নন-প্রাইম সংখ্যা $n (n > 1)$ এর অবশ্যই আরেকটা প্রাইম divisor p আছে, যেন $p \leq \sqrt{n}$ হয়।

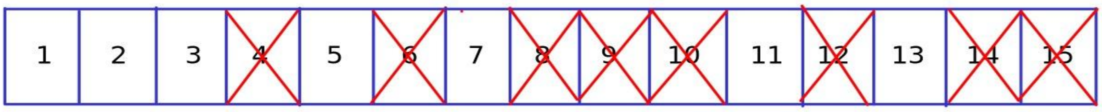
বুঝার সুবিধারতে, মনে করি N = 15।

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

আমরা এখানে 1 কে ignore করব! এখান আমরা জানি যে, 2 সবচেয়ে ছোট প্রাইম। অন্য কোন সংখ্যা যে 2 এর গুণিতক, প্রাইম হতে পারবে না। তাই ওদের আমরা কেটে দিব।

1	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15

এখন 2 এর পর next ফাঁকা ঘর হচ্ছে 3। এখানে 3 প্রাইম, কারণ অন্য কোন প্রাইম $p, (p < 3)$ যদি 3 এর divisor হতো, তাহলে 3 কাটা পরে যেত! তাই 3 এর সব গুণিতক কে কেটে দিব!



এভাবে চলতে থাকলে আমরা শেষ এ শুধু প্রাইম গুলো কাটা হবে না, বাকি সব সংখ্যা কাটা হয় যাবে! কারণ টা এবার দেখা যাক,

ধরে নিচ্ছি যে, আমরা এখান i এ আছি। $(1 < j < i)$ এমন সব প্রাইম j এর সব গুণিতক কে কাটা হয়ে গিয়েছে! এখন দুইটা জিনিস হতে পারে,

১) i এর ঘর কাটা হয়ে গেছে: এর অর্থাৎ কোন প্রাইম $p, (p < i)$ আছে, যেইটা i এর divisor। এইটা হলে i অবশ্যই প্রাইম না!

২) i এর ঘর কাটা হয়নাই: এইটার মানে, কোন প্রাইম $p, (p < i)$ নাই, যেইটা i এর divisor। এইটার মানে অবশ্যই i প্রাইম! এখান আমরা অন্য সব i এর গুণিতক কে কেটে দিব!

code টা দেখে ফেলা যাক!

```
vector < bool > mark(N+1, 1); // marking that no one has been cut yet!
vector < int > primes;

for(int i = 2; i <= N; i++){
    if(mark[i]){
        primes.push_back(i);
        for(ll j = 2*i; j <= N; j += i) mark[j] = 0;
    }
}
```

আগেই আমরা দেখেছি যে, এরকম সময়ে complexity $O(N \log N)$ হয়। কিন্তু এখানে যেহেতু শুধুমাত্র প্রাইম এর জন্য দিয়ে 2nd loop টা চলছে। তাই এ ক্ষেত্রে complexity $O(N \log \log N)$ হয়। এইটা এতই কম যে, আমরা $O(N)$ এই মনে করতে পারি!

তবে এটাকে $O(N \log \log \sqrt{N})$ এও করা যাবে। যদি j কে $2 * i$ থেকে শুরু করি তাহলে আমরা একই জিনিস বারবার করব। যেমন: $i = 2$ এর ক্ষেত্রে আমরা $2 \times 2, 2 \times 3, 2 \times 4, 2 \times 5 \cdots$ এ লুপ চালিয়েছি। $i = 3$ এর ক্ষেত্রে $j = 2 * i$ থেকে শুরু করলে আমরা $3 \times 2, 3 \times 3, 3 \times 4, 3 \times 5, \cdots$ এ লুপ চালাব। এখানে 2×3 তে আমরা দুইবার লুপ চালাচ্ছি। যেকোনো $k < i$ এর জন্যও $k \times i$ এ লুপ চালানো আগেই হয়ে যাবে। তাই আমরা $j = i \times i$ থেকে শুরু করব।

```
vector < bool > mark(N+1, 1); // marking that no one has been cut yet!
vector < int > primes;

for(int i = 2; i <= N; i++){
    if(mark[i]){
        primes.push_back(i);
        for(ll j = (long long)i*i; j <= N; j += i) mark[j] = 0;
    }
}
```

3 Modular Arithmetic

Nafis Ul Haque Shifat

অঙ্ক করার সময় আমরা ভাগশেষ নিয়ে খুব একটা মাথা না ঘামালেও গণিতের একটি আস্ত শাখাই আছে ভাগশেষ নিয়ে। হ্যাঁ, Modular Arithmetic হচ্ছে ভাগশেষের গণিত!
ধরা যাক, কোনো সংখ্যা n কে m দ্বারা ভাগ করলে ভাগশেষ থাকে r। তবে Modular Arithmetic এর ভাষায় লিখা যায়, $n \mod m = r$। অর্থাৎ $a \mod b$ কথাটির অর্থ হচ্ছে a কে b দ্বারা ভাগ করলে ভাগশেষ কত থাকে। যেমন $10 \mod 2 = 0$, $15 \mod 4 = 3$, $123 \mod 10 = 3$।

C++ এ ভাগশেষ নির্ণয়ের জন্য $\%$ অপারেটরটি ব্যাবহার করতে হয়।

```
cout << 5 % 3 << endl; // prints 2
cout << 120 % 10 << endl; // prints 0
```

3.1 Congruence

দুটি সংখ্যা x এবং y কে যদি m দ্বারা ভাগ করলে একই ভাগশেষ থাকে, তবে আমরা বলি $x \equiv y \pmod m$। এটিকে পড়া হয় x is congruent to y modulo m। যেমন $12 \equiv 7 \pmod 5$, কারণ 12 কে 5 দ্বারা ভাগ করলে ভাগশেষ থাকে 2, 7 কেও 5 দ্বারা ভাগ করলে ভাগশেষ থাকে 2। একই কারণে বলা যায়, $5 \equiv 1 \pmod 4$, $33 \equiv 0 \pmod 3$, $54 \equiv 12 \pmod 7$।
$x \equiv y \pmod m$ থেকে আরেকটি কথাও বুঝা যায়, তা হচ্ছে $(x - y)$, m এর একটি গুণিতক হবে, অর্থাৎ, $(x - y) \equiv 0 \pmod m$। কেন? লক্ষ্য কর x এবং y উভয়কেই m দ্বারা ভাগ করলে একই ভাগশেষ থাকে, ধরি তা হচ্ছে r। তবে লিখা যায়,

$$x = k_1 \times m + r$$

$$y = k_2 \times m + r$$

কাজেই,

$$(x - y) = (k_1 - k_2) \times m$$

দেখাই যাচ্ছে এবার আর কোনো ভাগশেষ বাকি নেই, কাজেই $(x - y) \equiv 0 \pmod m$।
একটি মজার ব্যাপার হচ্ছে, আমরা যদি কোনো সংখ্যা x এর সাথে m এর গুণিতক যোগ বা বিয়োগ দিতে থাকি, তবে ভাগশেষের কোনো পরিবর্তন হবে না! অর্থাৎ, $x \equiv x - m \equiv x - 2 \times m ... \equiv x + m \equiv x + 2 \times m .... \equiv x + k \times m \pmod m$।

3.2 Properties

ভাগশেষের কিছু চমৎকার বৈশিষ্ট্য রয়েছে, যার কারণেই Modular Arithmetic এত useful!

3.2.1 যোগ

ধরা যাক দুটি সংখ্যা a আর b এর যোগফল কে m দ্বারা ভাগ করলে কত ভাগশেষ থাকে তা বের করতে চাচ্ছি, অর্থাৎ $(a + b) \mod m = ?$ তা জানতে চাচ্ছি। মজার ব্যাপার হচ্ছে এর জন্য আমাদের কষ্ট করে

দুটি সংখ্যা যোগ করে তারপর ভাগশেষ নিতে হবে না, আমরা আলাদা করে $a \mod m$ ও $b \mod m$ নিয়ে তা যোগ করে দিলেই হবে! অর্থাৎ $a \mod m = r_1$ এবং $b \mod m = r_2$ হলে,

$$a + b \equiv r_1 + r_2 \pmod m$$

যেমন,

$$51 + 44 \equiv 2 + 2 \pmod 7$$
$$\implies 51 + 44 \equiv 4 \pmod 7$$

আচ্ছা এটি কেন কাজ করে? আগের মতই আমরা লিখতে পারি,

$$a = k_1 \times m + r_1$$

$$b = k_2 \times m + r_2$$

কাজেই,

$$a + b = (k_1 + k_2) \times m + (r_1 + r_2)$$

এখানে $(k_1 + k_2) \times m$ অংশটি m এর গুণিতক, অর্থাৎ $m \mid (k_1 + k_2) \times m$, কাজেই ভাগশেষ এর জন্য বাকি রইল $(r_1 + r_2)$ অংশটুকু!
এটি কি কাজে লাগে? একটা সমস্যা দেখা যাক!
Problem: দুটি পূর্ণ সংখ্যা n ও m দেওয়া হবে, যেখানে $n \le 10^6$, $m \le 10^9$, আমাদের n তম ফিবনাচ্চি সংখ্যা কে m দ্বারা ভাগ করলে কত ভাগশেষ থাকে তা বের করতে হবে।
খুব সহজ সমস্যা, আমরা জানি $f_i = f_{i-1} + f_{i-2}$, এটি দিয়ে একটি লুপ দিয়েই কাজটা করে ফেলা যায়! অনেকটা এভাবে-

```
int f[N];
f[0] = 0;
f[1] = 1;
for(int i = 2; i <= n; i++) {
    f[i] = f[i - 1] + f[i - 2];
}
cout << f[n] % m << endl;
```

কিন্তু এতে একটু খানি সমস্যা আছে, ফিবনাচ্চি ফাংশনটি n বাড়ার সাথে খুব দ্রুত বাড়তে থাকে, এতই দ্রুত যে n এর মান 100 ছোঁওয়ার আগেই f_n এতো বড় হয়ে যায় যে তা 64 bit integer এও জমা রাখা যায় না! তবে সমস্যা নেই, আমাদের f_n দরকার নেই, দরকার শুধু $f_n \mod m$। আমরা যদি array টিতে f_n না রেখে $f_n \mod m$ রেখে দেই, তবেই আর overflow হবে না। উত্তর এও গড়মিল হবার ভয় নেই, কারণ আমরা জানি $f_{n-1} + f_{n-2} \equiv ((f_{n-1} \mod m) + (f_{n-2} \mod m)) \pmod m$। এবার কাজটা এভাবে করা যায়,

```
int f[N];
f[0] = 0;
f[1] = 1;
for(int i = 2; i <= n; i++) {
    f[i] = (f[i - 1] + f[i - 2]) % m;
    //both f[i - 1] and f[i - 2] < m, so their sum won't overflow!
}
cout << f[n] << endl;
```

এভাবে প্রতি ধাপেই mod নিয়ে নিলে আর overflow নিয়ে মাথা ঘামাতে হয় না!

3.2.2 বিয়োগ

বিয়োগের নিয়মটিও যোগের মতই, অর্থাৎ $a \equiv r_1 \pmod m$ এবং $b \equiv r_2 \pmod m$ হলে,

$$a - b \equiv r_1 - r_2 \pmod m$$

যেমন,

$$50 - 31 \equiv 1 - 3 \pmod 7$$
$$\implies 50 - 31 \equiv -2 \pmod 7$$

আচ্ছা এখানে ঋণাত্মক সংখ্যা এলো কেন? ভুল হল নাকি? উত্তর তো আসার কথা ছিল 5, কেনোনা $50 - 31 = 19 \equiv 5 \pmod 7$।
আসলে ঠিক ই আছে, কারণ $-2 \equiv 5 \pmod 7$ । কিভাবে? আচ্ছা কোনো সংখ্যার সাথে তো 7 এর গুণিতক যোগ করলে তো $\bmod 7$ এ উত্তর পরিবর্তন হবে না, কেননা $7 \times k \equiv 0 \pmod 7$। কাজেই,

$$-2 \equiv -2 + 7 \pmod 7$$
$$\implies -2 \equiv 5 \pmod 7$$

কোড করার সময় $(a - b) \bmod m$ নির্ণয় করার সময় একটু সতর্ক থাকতে হয়, যেন ভাগশেষ টা 0 থেকে $m - 1$ এর মধ্যেই থাকে। এভাবে করা যেতে পারে -

```
int vagsesh = (a % m - b % m + m) % m;
cout << vagsesh << endl;
```

3.2.3 গুণ

আবার ও সেই একই নিয়ম, $a \equiv r_1 \pmod m$ এবং $b \equiv r_2 \pmod m$ হলে,

$$a \times b \equiv r_1 \times r_2 \pmod m$$

যেমন,

$$13 \times 24 \equiv 3 \times 4 \pmod{10}$$
$$\implies 13 \times 24 \equiv 12 \pmod{10}$$
$$\implies 13 \times 24 \equiv 2 \pmod{10}$$

এটির অনেক মজার একটি অ্যাপলিকেশন আছে, নিচের সমস্যা টা দেখা যাক।
Problem: a, n ও m দেওয়া আছে, আমাদের $a^n \bmod m$ নির্ণয় করতে হবে।
লক্ষ্য কর $a^n = a \times a \times \ldots \times a$ (n সংখ্যক), কাজেই গুনের নিয়ম দিয়ে একটি লুপ চালিয়েই আমরা এটি নির্ণয় করে ফেলতে পারি!

```
long long res = 1;
for(int i = 1; i <= n; i++) {
    res = (res * a) % m;
}
cout << res << endl;
```

আলাদা করে a^n নির্ণয় করে mod নিলে কিন্তু overflow হত, তাই প্রতি ধাপেই mod নিয়ে নিয়েছি। একটি লুপ লাগছে, তাই এর complexity $O(n)$। মজার ব্যাপার হচ্ছে এই কাজটি আর দ্রুত করা যায়, $O(\log n)$ complexity তেই $a^n \bmod m$ নির্ণয় করা সম্ভব! এটি খুব ই পরিচিত একটি টেকনিক, এর নাম হচ্ছে Big Mod!

3.2.4 Big Mod

এর আইডিয়া টা খুব সহজ, ধরা যাক 3^{40} নির্ণয় করতে চাচ্ছি। লক্ষ্য কর, $3^{40} = 3^{20} \times 3^{20}$। তার মানে আমরা যদি 3^{20} নির্ণয় করতে পারি তবে তা বর্গ করলেই 3^{40} পেয়ে যাব! আবার $3^{20} = 3^{10} \times 3^{10}$, কাজেই 3^{10} নির্ণয় করে তাকে বর্গ করে দিলেই 3^{20} ও পেয়ে যাব! একই ভাবে $3^{10} = 3^5 \times 3^5$, এবার 3^5 নির্ণয় করতে হবে। 5 তো বিজোড়, আগের মত করে দুভাগে ভাগ করতে পারব না, তবে আমরা এভাবে লিখতে পারি, $3^5 = 3 \times 3^4$, কাজেই 3^4 বের করে 3 দিয়ে গুণ দিলেই হচ্ছে!
আরেকটু গুছিয়ে বলার চেষ্টা করা যাক। ধরি $f(a, n)$ আমাদের a^n return করবে। যদি n জোড় হয় তবে লিখতে পারি $f(a, n) = f(a, \frac{n}{2}) \times f(a, \frac{n}{2})$। আর n বিজোড় হলে, $f(a, n) = a \times f(a, n-1)$। আমাদের যেহেতু a^n নয়, $a^n \mod m$ দরকার, প্রতি ধাপেই আমরা তাই mod নিতে থাকব!

$$f(a, n) = \begin{cases} 1 & n = 0 \\ a \times f(a, n-1) & n \mod 2 = 1 \\ f(a, \frac{n}{2}) \times f(a, \frac{n}{2}) & n \mod 2 = 0 \end{cases}$$

```
//ll = long long
ll bigmod(int a, int n) {
    if(n == 0) return 1; //base case

    if(n % 2 == 1) return a * bigmod(a, n - 1) % m;
    else {
        ll v = bigmod(a, n / 2);
        return v * v % m;
    }
}
```

এই অ্যালগরিদমটির এর complexity হচ্ছে $O(\log n)$।

3.2.5 ভাগ

ভাগের ক্ষেত্রে একটু ভেজাল আছে, এবার আর বাকি সব বারের মত $\frac{a}{b} \mod m = \frac{a \mod m}{b \mod m}$ নয়। $\frac{a}{b} \mod m$ বের করতে হলে a এর সাথে $b^{-1} \mod m$ গুণ করতে হবে। এখানে $b^{-1} \mod m$ কি? এটি হচ্ছে এমন একটি সংখ্যা, যার সাথে b গুণ করে mod m নিলে 1 পাওয়া যাবে, অর্থাৎ সংখ্যাটি x হলে $b \times x \equiv 1 \pmod{m}$। এখানে x কে m এর সাপেক্ষে b এর Modular Multiplicative Inverse বলা হয়! এমন x কি সবসময় পাওয়া যাবে? না, এমন x শুধু $gcd(b, m) = 1$ হলেই পাওয়া যাবে। এরকম x কিভাবে বের করতে হবে তা জানার আগে দু-একটি theorem জানা লাগবে। ধরে নেই x বের করে ফেলেছি কোনো ভাবে, তাহলে $\frac{a}{b} \equiv a \times x \pmod{m}$।

3.3 Theorems

3.3.1 Fermat's Little Theorem

p যদি কোনো একটি মৌলিক (prime) সংখ্যা হয় এবং a একটি পূর্ন সংখ্যা হয়, তবে $a^p \equiv a \pmod{p}$। এটি ই Fermat's Little Theorem! যেমন $2^{11} \equiv 2 \pmod{11}$।

Special Case: a যদি p দ্বারা বিভাজ্য না হয়, তবে Fermat's little theorem থেকে এটি বলা যায় যে, $a^{p-1} \equiv 1 \pmod{p}$। যেমন: $2^6 \equiv 1 \pmod{7}$

3.3.2 Euler's Theorem

Euler's Theorem বলে যে, n যদি একটি পূর্ণ সংখ্যা হয় এবং a এমন আরেকটি পূর্ণ সংখ্যা যেন a এবং n পরস্পর সহমৌলিক হয়, অর্থাৎ $gcd(a, n) = 1$ হয়, তবে,

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

এখানে $\phi(n)$ হচ্ছে Euler's Totient Function, এটি 1 থেকে n পর্যন্ত কয়টি সংখ্যা n এর সাথে সহমৌলিক তা বুঝায়। যেমনঃ $7^4 \equiv 1 \pmod{10}$, এখানে $\phi(10) = 4$।

3.3.3 Wilson's Theorem

Wilson's Theorem বলে যে, কোনো একটি পূর্ণ সংখ্যা n মৌলিক (prime) হবে যদি এবং কেবল যদি, $(n-1)! \equiv 1 \pmod{n}$ হয়।

3.4 Calculating Modular Inverse

একটু আগে যা বলেছিলাম আবার মনে করে নেই, কোনো একটি পূর্ণ সংখ্যা m এর সাপেক্ষে b এর Modular Inverse হচ্ছে এমন একটি সংখ্যা x যেন $b \times x \equiv 1 \pmod{m}$ হয়। এরকম হলে লিখা যায়, $b^{-1} \equiv x \pmod{m}$। আর এরকম x তখন ই পাওয়া যাবে যখন m এবং b সহমৌলিক হবে, অর্থাৎ $gcd(b, m) = 1$ হবে। m এর উপর ভিত্তি করে আমরা দুটি কেস এ ভাগ করে Modular Inverse নির্নয় করতে পারি।

1. m **মৌলিক সংখ্যা হলে:** আমরা Fermat's Little Theorem ব্যাবহার করতে পারি! যেহেতু $gcd(b, m) = 1$, তাই Fermat's Little theorem এর special case অনুসারে,

$$b^{m-1} \equiv 1 \pmod{m}$$
$$\implies b \times b^{m-2} \equiv 1 \pmod{m}$$

সমীকরণটার দিকে আবার তাকাও, লক্ষ্য কর, b ও b^{m-2} এর গুণফলের সাথে $mod\ m$ নিলে 1 পাওয়া যাচ্ছে! Modular Inverse এর সংজ্ঞা এর সাথে b^{m-2} পুরোপুরি মিলে গিয়েছে! কাজেই b^{m-2} নিশ্চয়ই b এর Modular Inverse! অর্থাৎ $b^{-1} \equiv b^{m-2} \pmod{m}$। এবার $b^{m-2} \mod m$ তো আমরা Big Mod দিয়েই বের করে ফেলতে পারি!

2. m **মৌলিক সংখ্যা না হলে:** এবার Euler's Theorem কাজে লাগাব। আমরা জানি,

$$b^{\phi(m)} \equiv 1 \pmod{m}$$
$$\implies b \times b^{\phi(m)-1} \equiv 1 \pmod{m}$$

বুঝাই যাচ্ছে, $b^{\phi(m)-1}$ হচ্ছে b এর Modular Inverse, অর্থাৎ $b^{-1} \equiv b^{\phi(m)-1} \pmod{m}$। সমস্যা হচ্ছে আমাদের এবার $\phi(m)$ বের করতে হবে, তবে কাজটা বেশি কঠিন নয়, একটি খুব সহজ সূত্রই আছে! যদি $m = p_1^{a_1} \times p_2^{a_2} \times ... \times p_k^{a_k}$ হয়, তবে,

$$\phi(m) = m \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times ... \times \left(1 - \frac{1}{p_k}\right)$$

$\phi(m)$ বের করতে পারলে আবার ও আমরা Big Mod দিয়েই বাকি কাজ করে ফেলতে পারব!

3.5 Problems

শেষ করার আগে চিন্তা করার জন্য কিছু প্রব্লেম দেয়া যাক।

1. সবাই হয়তো জানো যে, কোনো সংখ্যা n, 9 দ্বারা বিভাজ্য হলে n এর অঙ্ক গুলো ও 9 দ্বারা বিভাজ্য হয়! যেমন $9 \mid 126$, আবার $9 \mid (1+2+6)$। এটি সবসময় সত্য, কেন তা বলতে পারবে? কথাটি কি 3 এর জন্য ও সত্যি?

2. শুধু হাতে কলমে 3^{28} এর শেষ অংক কত তা বের করতে পারবে?

3. তোমাকে একটি বিশাআআল বড় সংখ্যা x দেয়া আছে, তাতে সর্বোচ্চ 10^6 টা পর্যন্ত অংক থাকতে পারে! সাথে আরেকটি সংখ্যা m ও দেয়া আছে, তোমাকে $x \mod m$ বের করতে হবে।