素数、约数与筛法

试除法分解 + 埃氏筛 / 欧拉线性筛 $O(n)$ + 区间筛法

HGNU-ACM 实验室算法组

零基础精讲：$\sqrt{n}$ 试除与唯一分解 + 最小质因子不重筛 + 2 道经典例题逐行推导





本章目录

试除法判素数与质因数分解

埃氏筛与欧拉线性筛 $O(n)$

超大值域 $[L, R]$ 的区间筛法

第 1 节用 $\sqrt{n}$ 对称性学会判素数与质因数分解；第 2 节掌握埃氏筛与 $O(n)$ 欧拉线性筛；第 3 节攻克 $R \leq 2^{31}$ 的区间筛法。

本章目录（续）

例题精讲一：质因数分解（一本通 1620）

例题精讲二：Prime Distance（一本通 1619）

筛法实战：哥德巴赫猜想验证

第 4∼5 节精讲《质因数分解》《Prime Distance》两道分层例题；第 6 节汇总高频错误、检查清单与练习安排。

承上启下与生活引入：把 12 块糖摆成长方形

- 上一章《快速幂、矩阵与线性递推》带我们把指数运算加速到了 $O(\log b)$。从本章开始，我们走进整数的“内部结构”——素数与约数。
- 先看一个生活小例子：手上有 12 块糖，要摆成整整齐齐的长方形，有几种摆法？
  - 1×12、2×6、3×4（再往下 4×3、6×2、12×1 只是把长和宽反过来）。
- 如果手里是 7 块糖呢？只能摆成 1×7 一条长队，怎么也凑不出别的长方形！

直觉

约数总是成对出现的；而 7 这种只能摆成 1×7 的数，就是不能再拆分的“整数积木”——素数！

本页要点

找约数只需要找“较小的一边”（长 $\leq$ 宽），另一边自动通过除法 n/i 得到！

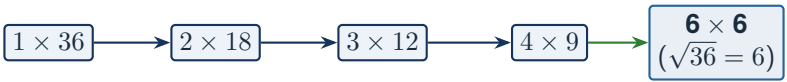
新术语大白话：素数、合数、约数与为什么 1 不是素数

- 约数（因数）：如果整数 n 除以正整数 d 没有余数（`n % d == 0`），就说 d 是 n 的约数。
- 素数（质数）：大于 1 的整数中，只有 1 和它自己两个正约数的数（如 $2, 3, 5, 7, 11$）。其中 2 是唯一的偶素数。
- 合数：大于 1 且除了 1 和它自己之外还有其他约数的数（如 $4, 6, 9, 12$）。
- 为什么 1 既不是素数也不是合数？
  - 如果把 1 算成素数，$6 = 2 \times 3 = 1 \times 2 \times 3 = 1 \times 1 \times 2 \times 3$，拆法就不唯一了！为了保住“唯一分解”，数学界规定 1 不是素数。

本页要点

做素数题的第一条铁律：$n \leq 1$ 一律不是素数！写判素数函数必须先写 `if (n < 2) return false;`。

为什么试除到 $\sqrt{n}$ 就够了？对称性图解



- 如果合数 $n = a \cdot b$ 且两个因子都大于 $\sqrt{n}$，那么 $a \cdot b > \sqrt{n} \cdot \sqrt{n} = n$，矛盾！
- 因此任意合数 n 必定至少有一个因子 $\leq \sqrt{n}$。只要在 $[2, \lfloor \sqrt{n} \rfloor]$ 内找不到任何约数，n 就一定是素数！

本页要点

用 `i <= n / i`（或 `1LL * i * i <= n`）作为循环上限，千万别写 `i * i <= n` 防止 `int` 溢出！

暴力 $O(n)$ 试除 vs $O(\sqrt{n})$ 试除：操作次数对比

- 暴力做法：从 2 枚举到 $n-1$，看有没有能整除 n 的数。
- 算算操作次数（假设 $n = 2 \times 10^9$ 且 n 是素数）：
  - $O(n)$ 暴力：要循环 2×10^9 次，耗时约 $10 \sim 20$ 秒，单次判断直接 **TLE**！
  - $O(\sqrt{n})$ 试除：只需循环 $\lfloor \sqrt{2 \times 10^9} \rfloor \approx 44721$ 次，耗时不到 0.1 毫秒！

边界提醒

如果用 `int i` 写 `for (int i = 2; i * i <= n; ++i)`，当 $n = 2147483647$ 时，$i = 46341$ 的平方 $46341^2 = 2147488281$ 会超出 `int` 上限溢出成负数，导致死循环或 RE！

本页要点

写 `i <= n / i` 是最安全的写法：除法绝不会溢出，还能准确覆盖 $\lfloor \sqrt{n} \rfloor$！

写判素数前，先回答三个问题

把重复试除想清楚再写代码

想判断 $n = 29$ 是不是素数，我们需要用循环试除可能成为因子的数 i：

你要决定的问题	试除法里怎么选
1. 边界 $n \le 1$ 怎么办？	进循环前直接特判 `if (n < 2) return false;`
2. i 从哪里开始、到哪里停？	从 $i = 2$ 开始，只要 `i <= n / i` 就继续试
3. 找到整除就怎样、全没找到怎样？	一旦 `n % i == 0` 立刻返回 `false`；循环走完返回 `true`

本页要点

先特判 $n < 2$，再写安全上限 `i <= n / i`，能避开 90% 的试除法低级错误！

手算跟踪表：用 $O(\sqrt{n})$ 试除判定 $n = 29$ 与 $n = 25$

待测 n	当前 i	条件 $i \le n/i$	`n % i == 0`?	结果与动作
29	2	$2 \le 14$（真）	$29 \bmod 2 = 1 \neq 0$	继续，`i++`
29	3	$3 \le 9$（真）	$29 \bmod 3 = 2 \neq 0$	继续，`i++`
29	4	$4 \le 7$（真）	$29 \bmod 4 = 1 \neq 0$	继续，`i++`
29	5	$5 \le 5$（真）	$29 \bmod 5 = 4 \neq 0$	继续，`i++`
29	6	$6 \le 4$（假）	终止检查	退出循环，返回 **true**（素数）
25	$2 \sim 4$	真	均不整除	依次 `i++`
25	5	$5 \le 5$（真）	$25 \bmod 5 = 0$	立即返回 **false**（合数）

本页要点

注意 $n = 25$ 时，$i = 5$ 恰好满足 $5 \le 25/5$，所以一定要写 `<=` 而不能写 `<`!

试除法判素数模板代码与为什么对

```cpp
bool is_prime(long long n) {
    if (n < 2) return false;                  // 0、1 与负数都不是素数
    for (long long i = 2; i <= n / i; ++i) { // 等价于 i*i <= n，但绝不溢出
        if (n % i == 0) return false;         // 找到一个 [2, sqrt(n)] 内的因子，必为合数
    }
    return true;                              // 一个因子都没找到，必为素数
}
```

正确性（循环不变量）

- 保持：每轮开始前，$[2, i-1]$ 中没有任何 n 的约数。
- 终止：若某步 $n \bmod i == 0$ 则返回 `false`；若循环正常结束，则 $i > \sqrt{n}$，由对称性 $[2, n-1]$ 均无约数，故 n 为素数。

本页要点

单次判素数只需 6 行，时间复杂度 $O(\sqrt{n})$，空间复杂度 $O(1)$。

小优化：为什么 2 以外的偶数都不用试？

- 如果 n 不是 2 的倍数，那么 $4, 6, 8, 10 \ldots$ 这些偶数也绝对不可能整除 n！
- 因此，先单独检查 $n == 2$（返回 `true`）和 `n % 2 == 0`（返回 `false`），之后循环从 $i = 3$ 开始，每次 `i += 2` 只试奇数！
- 这样操作次数直接减半，从 $\sqrt{n}$ 降到 $\frac{1}{2}\sqrt{n}$。

本页要点

在复杂度仍然是 $O(\sqrt{n})$ 的情况下，只试奇数能把单次判素数的常数直接砍半！

新术语大白话：算术基本定理（唯一分解定理）

- 算术基本定理：任何一个大于 1 的整数 n，如果不计质因子的排列顺序，都可以唯一地拆成若干个质数的乘积：

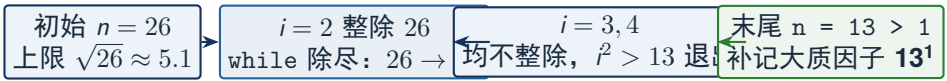
$$n = p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k} \quad (p_1 < p_2 < \cdots < p_k \text{ 均为质数})$$

- 大白话理解：每个合数都是由“质数积木”拼成的，而且每个数只有唯一一张配方表。例如：$12 = 2^2 \times 3^1$，$84 = 2^2 \times 3^1 \times 7^1$。
- 为什么分解时只用除到 $\sqrt{n}$?
  - 在一个正整数 n 的所有质因子中，最多只能有 1 个质因子大于 $\sqrt{n}$!（如果有两个 $> \sqrt{n}$ 的质因子，它们乘起来就已经大于 n 了!）

本页要点

把 $\leq \sqrt{n}$ 的质因子全部除干净之后，如果剩下的 `n > 1`，它一定就是那个唯一的大质因子!

图解质因数分解：为什么末尾必须检查 if (n > 1)？



- 当我们用 `for (long long i = 2; i <= n / i; ++i)` 分解时，一旦把 2 除掉，n 缩成了 13，循环在 $i = 4$ 就退出了！
- 如果漏掉末尾的 `if (n > 1)`，26 就会被错分解成只有 2，丢掉了 13！

本页要点

每除掉一个质因子就让 `n /= i`，不仅缩小了剩余规模，也让 `i <= n / i` 提前结束，速度飞快！

手算跟踪表：质因数分解 $n = 84$ 与 $n = 26$

待分解数	当前 i	条件 $i \le n/i$	动作	剩余 n 与已得质因子
84	2	$2 \le 42$（真）	`while (n%2==0)` 除两次	$n: 84 \to 42 \to 21$，记录 2^2
21	3	$3 \le 7$（真）	`while (n%3==0)` 除一次	$n: 21 \to 7$，记录 3^1
7	4	$4 \le 1$（假）	退出循环	此时 $n = 7$
7	末尾检查	`n > 1`（真）	补记剩余 n	记录 7^1，得 $84 = 2^2 \times 3^1 \times 7^1$
26	2	$2 \le 13$（真）	除一次	$n: 26 \to 13$，记录 2^1
13	3	$3 \le 4$（真）	不整除	$n = 13$，`i++`
13	4	$4 \le 3$（假）	退出并检查 `n > 1`	补记 13^1，得 $26 = 2^1 \times 13^1$

本页要点

你可能会问：为什么 `i` 枚举到 4（合数）时不会把合数当质因子？因为 4 的质因子 2 早在 $i = 2$ 时就被除干净了！

质因数分解模板代码与上机验证

```
void factorize(long long n) {
    for (long long i = 2; i <= n / i; ++i) {
        if (n % i == 0) {
            int cnt = 0;
            while (n % i == 0) { n /= i; ++cnt; } // 把当前质因子 i 彻底除尽
            cout << i << "^" << cnt << " ";
        }
    }
    if (n > 1) cout << n << "^1";           // 别忘了可能剩下的唯一大质因子!
    cout << '\n';
}
```

- **3** 组上机验证：输入 12 → 2^2 3^1；输入 13 → 13^1；输入 1 → 空行（无质因子）。
- 反事实追问：删掉 `if (n > 1)` 后输入 14 会怎样？（只输出 2^1，丢掉 7^1！）

本页要点

用 `while (n % i == 0)` 除尽较小质因子，既统计指数 `cnt`，又保证后续因子必为质数。

约数个数定理、约数和定理与求所有约数

- 设 $n = p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k}$，由乘法原理可得两大高频定理：
  1. 约数个数定理：$d(n) = (c_1 + 1)(c_2 + 1) \cdots (c_k + 1)$（每个质因子 p_i 可选 $0 \sim c_i$ 个，共 $c_i + 1$ 种选法）。
  2. 约数和定理：$\sigma(n) = \prod_{i=1}^{k} (1 + p_i + p_i^2 + \cdots + p_i^{c_i})$。
- 手算验证 $n = 12 = 2^2 \times 3^1$：
  - $d(12) = (2 + 1)(1 + 1) = 6$ 个（分别是 $1, 2, 3, 4, 6, 12$）。
  - $\sigma(12) = (1 + 2 + 4)(1 + 3) = 7 \times 4 = 28$（验证：$1 + 2 + 3 + 4 + 6 + 12 = 28$）。
- 直接收集所有约数：从 $i = 1$ 枚举到 $i \le n/i$，若 $n \bmod i == 0$ 就收下 i；若 $i \neq n/i$ 再收下 n/i（防止完全平方数重复加入 $\sqrt{n}$），最后 `sort`。

本页要点

乘法原理把“枚举所有约数”变成了只看质因子指数的连乘，这是《反素数》《聪明的燕姿》的核心武器。

手算小练习：用约数公式计算 $n = 72$ 的约数个数与约数和

- 先把 72 分解质因数：$72 = 8 \times 9 = 2^3 \times 3^2$。
- 算约数个数：$d(72) = (3+1) \times (2+1) = 4 \times 3 = \mathbf{12}$ 个。
  - 为什么是乘法？选 2 的幂次有 $\{2^0, 2^1, 2^2, 2^3\}$ 共 4 种选法，选 3 的幂次有 $\{3^0, 3^1, 3^2\}$ 共 3 种选法，自由搭配共 $4 \times 3 = 12$ 种！
- 算约数和：$\sigma(72) = (1+2+4+8) \times (1+3+9) = 15 \times 13 = \mathbf{195}$。

本页要点

记住“把展开式乘开就是所有约数相加”，你就永远不会忘记约数和公式 $\sigma(n)$ 的写法！

模板补充：用试除法收集一个数的所有正约数

```cpp
vector<long long> get_divisors(long long n) {
    vector<long long> divs;
    for (long long i = 1; i <= n / i; ++i) {
        if (n % i == 0) {
            divs.push_back(i);              // 收集较小一侧的约数
            if (i != n / i) divs.push_back(n / i); // 避免完全平方数重复收集
        }
    }
    sort(divs.begin(), divs.end());         // 若需要有序输出则排序
    return divs;
}
```

本页要点

以 $n = 36$ 为例，当 $i = 6$ 时 n/i 也是 6，加上 `if (i != n / i)` 才能保证 6 只被放进数组一次！

试除法常见错误对照表

错误写法	为什么错	正确写法与评测结果
判素数漏写 `n < 2`	输入 1 或 0 时循环进不去，误判为素数	开头先写 `if (n < 2) return false;`（避免 **WA**）
写 `int i; i*i <= n`	n 接近 $2^{31}-1$ 时 `i*i` 溢出变负	改写 `i <= n / i`（避免 **TLE/WA**）
分解末尾漏 `if (n > 1)`	丢失大于 $\sqrt{n}$ 的最大质因子	循环后补判 `if (n > 1)`（避免 **WA**）
求约数没判 `i != n / i`	完全平方数的 $\sqrt{n}$ 被重复放进数组两次	加 `if (i != n / i)` 再收 `n/i`（避免 **WA**）

本页要点

如果只需要判断/分解少数几个大数（$n \le 10^{12}$），试除法 $O(\sqrt{n})$ 足够；但要一次查询 $1 \sim 10^7$ 所有素数怎么办？

生活引入与为什么需要筛法

- 新需求：给定 $N = 10^7$，要找出 $1 \sim N$ 里的所有素数（或者回答 10^6 次“x 是不是素数”）。
- 算算试除法操作次数：对 $1 \sim 10^7$ 每个数都跑一次 $O(\sqrt{n})$ 试除，总运算量约 $\sum_{i=1}^{10^7} \sqrt{i} \approx 2 \times 10^{10}$ 次——肯定 **TLE**！
- 生活场景“划名单”：与其一个个去检验谁是素数，不如把 $2 \sim N$ 排成一排，每遇到一个没被划掉的数（它一定是素数），就顺手把它的倍数全部划掉！

角色反转

试除法是“被动挨个检查”；筛法是“主动把合数全部划掉”，没被划掉的自动就是素数！

本页要点

当需要批量求 $1 \sim N$ 的素数表（$N \leq 10^7$）时，必须放弃逐个试除，改用筛法！

埃氏筛核心思路与独占图解（$N = 15$）



- 括号里写的是“谁划掉了它”。可以看到 6 和 12 都被 2 和 3 重复划了两遍！
- 为什么可以从 p^2 开始划？因为 $2p, 3p, \ldots, (p-1)p$ 一定含有比 p 更小的质因子，早在前面就已经被划过了！

本页要点

埃氏筛的唯一美中不足是“合数会被多个质因子重复划掉”，比如 30 会被 $2, 3, 5$ 划三遍！

埃氏筛手算跟踪表：手工追踪 $N = 15$ 的划数过程

当前 i	comp[i]==0?	内层 j 从 i^2 起划掉的数	备注（观察重复）
2	是（素数）	$4, 6, 8, 10, 12, 14$	划掉所有 ≥ 4 的偶数
3	是（素数）	9, **12**, 15	注意 **12** 被 2 和 3 划了两次！
4	否（已划）	跳过	$4^2 = 16 > 15$，外层也将结束

本页要点

外层只需枚举到 $\lfloor \sqrt{15} \rfloor = 3$；最后 comp[2..15] 中仍为 0 的就是 $2, 3, 5, 7, 11, 13$。

埃氏筛模板代码与 $O(n \log \log n)$ 复杂度

```
vector<char> comp(N + 1, 0);                    // comp[x] = 1 表示合数，0 表示素数
comp[0] = comp[1] = 1;                          // 0 和 1 都不是素数
for (int i = 2; 1LL * i * i <= N; ++i) {        // 只需用 <= sqrt(N) 的质数去筛
    if (!comp[i]) {
        for (int j = i * i; j <= N; j += i) // 从 i*i 开始步长 i 划掉倍数
            comp[j] = 1;
    }
}
```

- 为什么复杂度是 $O(n \log \log n)$？每个素数 $p \le \sqrt{N}$ 大约划掉 N/p 个数，总操作数约为 $N \sum_{p \le N} \frac{1}{p} \approx N \ln \ln N$。当 $N = 10^7$ 时 $\ln \ln N \approx 2.8$，已经非常快！
- 注意防溢出：外层循环条件用 `1LL * i * i <= N`，否则 N 较大时 `i*i` 可能溢出。

本页要点

埃氏筛代码只有 7 行、极好写，在区间筛预处理和 $N \le 10^7$ 仅求素数表时完全够用！

埃氏筛上机验证与反事实追问

- **3** 组上机验证（统计 $2 \sim N$ 中素数个数）：
  1. $N = 1 \to 0$（因为提前设了 `comp[0]=comp[1]=1`）；
  2. $N = 15 \to 6$；
  3. $N = 10000000$（一千万）$\to$ `664579`（耗时约 0.08 秒）。

反事实追问

如果把内层起点 `j = i * i` 改成 `j = 2 * i`，答案还对吗？为什么不建议这样写？答案依然全对，但每个素数都要多划前面已经被更小质因子划掉的数，常数变慢且外层必须跑完 $2 \sim N$ 而不能停在 $\sqrt{N}$。

本页要点

用 `j = i * i` 既减少了划数次数，又让外层循环在 $\sqrt{N}$ 处提前收工！

写线性筛前，先回答三个问题

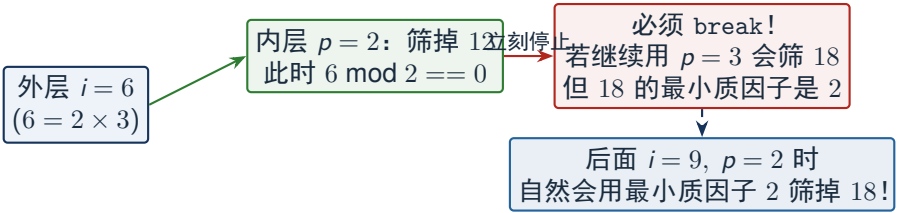
为了彻底消灭埃氏筛的重复划数，欧拉提出了一条铁律：每个合数只被它自己的最小质因子筛掉一次！

写之前先回答	欧拉线性筛里怎么设计
1. 用什么记录已经找到的质数？	用数组 `primes` 按从小到大的顺序存质数
2. 怎么让每个合数只被最小质因子筛？	外层枚举倍数 `i`，内层枚举已发现的质数 `p`，筛掉 `i * p`
3. 什么时候必须立刻 `break`？	当 `i % p == 0` 时立刻停止内层循环！

本页要点

线性筛的外层 `i` 既是“待判定的数”，也是“将来用来乘出更大合数的系数”！

图解为什么必须 `if (i % p == 0) break;`（以 $i = 6$ 为例）



- 核心原因：一旦 `i % p == 0`，说明 p 是 i 的一个质因子（而且因为 `primes` 从小到大遍历，p 就是 i 的最小质因子）。
- 若再往后拿更大的质数 $p' > p$ 去乘 i，得到的合数 $i \cdot p'$ 依然含有因子 p，它的最小质因子是 p 而不是 p'！

本页要点

一句话记住：`i % p == 0` 说明“再往后 p' 就抢了 p 的最小质因子饭碗”，必须 `break`！

手算跟踪表：手工追踪 $N = 12$ 的欧拉线性筛

i	质数?	primes	内层 p 筛数与 break 原因
2	是（质数）	$[2]$	$p = 2$: 筛 4（$2 \bmod 2 == 0$ **break**）
3	是（质数）	$[2, 3]$	$p = 2$: 筛 6；$p = 3$: 筛 9（$3 \bmod 3 == 0$ **break**）
4	否（$minp = 2$）	$[2, 3]$	$p = 2$: 筛 8（$4 \bmod 2 == 0$ **break**，不筛 12）
5	是（质数）	$[2, 3, 5]$	$p = 2$: 筛 10；$p = 3$: $5 \times 3 = 15 > 12$ 越界停
6	否（$minp = 2$）	$[2, 3, 5]$	$p = 2$: 筛 **12**（$6 \bmod 2 == 0$ **break**）
$7 \sim 12$	依次判定	$[2, 3, 5, 7, 11]$	终止：$i = 13 > 12$ 退出循环

本页要点

看 12：在 $i = 4, p = 3$ 时被提前 break 躲过，最终只在 $i = 6, p = 2$ 时被唯一筛了一次！

为什么对？证明每个合数 x 恰好被筛一次

设任意合数 $x \le N$ 的最小质因子为 p，令 $i = x/p$。

证明维度	为什么一定成立
1. 会被筛到（存在性）	因为 p 是 x 的最小质因子，所以 $i = x/p$ 的所有质因子都 $\ge p$。当外层走到 i 时，内层遍历到比 p 小的质数都不会整除 i，因此不会提前 `break`，必能走到 p 并筛掉 $x = i \cdot p$！
2. 只筛一次（唯一性）	假设 x 还能被另一个质数 $q > p$ 在 $i' = x/q$ 时筛掉。但 $p \mid x$ 且 $p \neq q$，故 $p \mid i'$！于是内层在走到 q 之前就先遇到 p 并触发 `i' % p == 0` 而 `break`，不可能走到 q！

本页要点

存在性 + 唯一性 = 每个合数精确被赋值一次，总操作次数严格线性 $O(n)$！

欧拉线性筛标准模板（顺带记录最小质因子 minp[]）

```
vector<int> primes, minp(N + 1, 0);
for (int i = 2; i <= N; ++i) {
    if (!minp[i]) { minp[i] = i; primes.push_back(i); } // 没被更小质因子筛掉，必为素数
    for (int p : primes) {
        if (1LL * i * p > N) break;          // 超出范围，直接结束内层
        minp[i * p] = p;                     // 无论是否 break，i*p 的最小质因子都是 p
        if (i % p == 0) break;               // p 是 i 的最小质因子，再往后就重复了
    }
}
```

- 写筛技巧：用 `minp[i] == 0` 兼任“是不是合数”的标记，一举两得，省去一个布尔数组！
- 内存提醒：若只需判素数，用 `vector<char>` 或静态 `bool[]`，避免 `vector<bool>` 位打包开销。

本页要点

`minp[i*p] = p;` 必须写在 `if (i%p == 0) break;` 前面，否则 $12 = 6 \times 2$ 还没记上就退出了！

白送福利：用 minp[x] 实现 $O(\log x)$ 极速质因数分解

- 有了预处理好的最小质因子数组 minp[]，分解任意 $x \le N$ 时不再需要从 2 试除到 $\sqrt{x}$，只要每次直接除以 minp[x]！

```
while (x > 1) {
    int p = minp[x];                          // 直接 O(1) 查出当前最小质因子
    int cnt = 0;
    while (x % p == 0) { x /= p; ++cnt; }     // 除尽这个质因子
    cout << p << "^" << cnt << " ";
}
```

- 手算 $x = 60$：minp[60]=2 除两次剩 15 → minp[15]=3 除一次剩 5 → minp[5]=5 除一次剩 1，仅 3 步！

本页要点

当题目需要把 10^6 个不超过 10^7 的数分别分解质因数时，minp[] 方案比逐个 $O(\sqrt{x})$ 快上千倍！

手算跟踪表：用 minp[x] 极速分解 $x = 60$

假设线性筛已经算出 $\text{minp}[60] = 2,\ \text{minp}[30] = 2,\ \text{minp}[15] = 3,\ \text{minp}[5] = 5$：

当前 x	查表 $p = \text{minp}[x]$	内层 `while (x % p == 0)`	输出与轮末 x
60	2	除两次：$60 \to 30 \to 15$	输出 2^2，x 变为 15
15	3	除一次：$15 \to 5$	输出 3^1，x 变为 5
5	5	除一次：$5 \to 1$	输出 5^1，x 变为 1
终止	—	`x > 1` 为假	退出循环，完成分解！

本页要点

每次直接跳到最小质因子，不需要任何无效试除，分解 60 只查了 3 次表！

线性筛预告：顺带 $O(n)$ 求出所有积性函数（如欧拉函数 φ）

- 因为欧拉线性筛保证每个合数 $x = i \cdot p$ 只被生成一次，且我们还能精准区分“p 整除 i”和“p 不整除 i”两种情况，所以可以在线性筛的同时顺手算出：
  - 欧拉函数 $\varphi(n)$（$1 \sim n$ 中与 n 互质的数的个数，第 3 章重点）；
  - 约数个数 $d(n)$ 与约数和 $\sigma(n)$；
  - 莫比乌斯函数 $\mu(n)$。

本页要点

这就是为什么算法竞赛常说“学会欧拉线性筛，就拿到了数论积性函数的万能钥匙”。

上机验证、反事实追问与两大筛法对比

- **3** 组上机验证（输出 N 以内质数个数）：
  1. $N = 1 \to 0$（最小边界）；
  2. $N = 15 \to 6$（分别是 $2, 3, 5, 7, 11, 13$）；
  3. $N = 1000000 \to 78498$（一百万以内恰好有 78498 个质数）。
- 反事实追问：如果把 `if (i % p == 0) break;` 删掉，程序答案还会对吗？耗时怎么变？（答案依然对，但每个合数会被重复赋值多次，时间退化且略慢于埃氏筛。）

筛法	时间复杂度	代码长度	适用场景
埃氏筛	$O(n \log \log n)$	7 行	只需要判素数、区间筛预处理小质数
欧拉线性筛	$O(n)$	9 行	$N \geq 10^7$、需顺带求 `minp[]` / 欧拉函数 / 莫比乌斯函数

本页要点

两者的空间复杂度都是 $O(n)$；如果值域 R 高达 $2^{31} - 1$，普通数组装不下怎么办？看下一节区间筛！

筛法常见错误对照表

错误写法	为什么错	正确写法与评测结果
`break` 写在 `minp[i*p]=p` 前	当 $i\%p == 0$ 时 $i \cdot p$ 本身没被筛掉	先赋值 `minp[i*p]=p` 再 `break`（避免 **WA**）
判断越界写 `i * p <= N`	`int` 相乘可能先溢出成负数	写成 `1LL * i * p <= N`（避免 **RE/WA**）
开 `int minp[1e8]`	10^8 个 `int` 占 400 MB，超内存	超出 256 MB 限制必 **MLE**，按需开大小
错用 `vector<bool>`	位运算打包在超大范围较慢	优先用 `vector<char>` 或静态 `bool[]`

本页要点

开数组前先算内存：10^7 个 `int` 约 40 MB，10^7 个 `char/bool` 仅 10 MB。

问题引入与核心洞察：当 R 达到 $2^{31}-1$ 时怎么办？

- 遇到难题：给定区间 $[L,R]$，其中 $1 \le L < R \le 2^{31}-1$（约 21.4 亿），但区间长度 $R-L \le 10^6$。要求找出 $[L,R]$ 里的所有素数！
- 为什么普通筛法直接挂掉？开一个长度为 2.14×10^9 的 `bool` 数组需要 2.14 GB 内存（直接 **MLE**），遍历一遍也要 21 亿次操作（直接 **TLE**）！
- 核心洞察（两大武器联手）：
  1. $[L,R]$ 里任意一个合数，都必定有一个不超过 $\sqrt{R} \le \lfloor \sqrt{2^{31}} \rfloor = 46341$ 的质因子！
  2. 虽然 R 很大，但区间长度只有 10^6，我们完全可以开一个长度为 $R-L+1$ 的偏移数组 `bad[]`，用下标 j 代表数字 $L+j$！

本页要点

把“在 $[1,21$亿$]$ 里找素数”转化为“用 ≤ 46341 的小质数去筛长度为 10^6 的滑动窗口”！

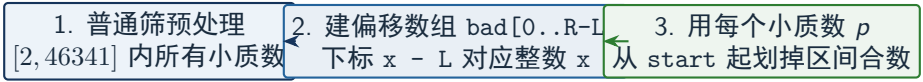
为什么区间筛只需要预处理到 $\sqrt{R}$？（反证法 + 实例）

- 反证法：假设区间 $[L, R]$ 里有一个合数 x，它没有任何 $\le \sqrt{R}$ 的质因子。
- 那么 x 至少可以拆成两个大于 $\sqrt{R}$ 的因子之积：$x = a \cdot b > \sqrt{R} \cdot \sqrt{R} = R$，这与 $x \le R$ 矛盾！
- 具体数字例子：当 $R = 2^{31} - 1 = 2147483647$ 时，$\sqrt{R} \approx 46340.95$。
  - 因此我们只需要把 $[2, 46341]$ 里的 4792 个小质数筛出来，就能把 $[1, 2^{31} - 1]$ 里的所有合数全部筛光！

本页要点

用 4792 个小质数就能统治 21 亿范围的合数判定，这就是 $\sqrt{R}$ 上界的威力！

图解区间筛法三步走



- 第一步只需筛到 46341，耗时不到 1 毫秒！
- 第二步特判：`if (L == 1) bad[0] = 1;`（因为 1 不是素数，而任何质数的倍数都不会划掉 1!）。

本页要点

偏移映射 `x <-> x - L` 是把百亿级坐标压进 10^6 数组的万能技巧。

关键公式推导：为什么起点必须取 max(p * p, ...)？

对每个小质数 p，我们要找到它在 $[L, R]$ 内的第一个倍数作为起点 start：

- 大于等于 L 的最小 p 的倍数是：

$$\left\lceil \frac{L}{p} \right\rceil \cdot p = \frac{L + p - 1}{p} \cdot p \quad \text{(C++ 整数除法)}$$

- 为什么不能直接用它，还要和 p^2 取 max？
  - 假设 $L = 2,\ R = 10$，当前小质数 $p = 3$。
  - 只算向上取整：$(2 + 3 - 1)/3 \times 3 = 3$——直接把质数 3 本身标记成了合数（自杀）！
  - 取 start = max(p * p, (L + p - 1) / p * p) 后，起点变成 $\max(9, 3) = 9$，完美跳过 3 和已被 2 筛过的 6！

本页要点

只有合数才该被划掉；max(p * p, ...) 保证起点至少是 p^2，永远不会误杀质数 p 自己！

手算跟踪表：手工追踪区间 $[1, 10]$ 的区间筛

预处理 $\leq \sqrt{10}$ 的小质数：$p \in \{2, 3\}$；建数组 `bad[0..9]`（对应 $1 \sim 10$），初值全 0：

步骤	操作或公式计算	标记结果（对应原数）
特判 $L = 1$	`if (L == 1) bad[0] = 1;`	划掉 1
$p = 2$	`start = max(4, (1+1)/2*2) = 4`	划掉 $4, 6, 8, 10$（保留了素数 2!）
$p = 3$	`start = max(9, (1+2)/3*3) = 9`	划掉 9（保留了素数 3!）
扫描输出	枚举 $x = 1 \ldots 10$ 检查 `!bad[x - 1]`	剩下的为 **2**, **3**, **5**, **7**

本页要点

看 $[1, 10]$ 这个最小边界：$L = 1$ 特判和 `max(p*p, ...)` 两个细节同时发挥了作用！

区间筛法标准模板（12 行）

```
// 已知 primes 存有 <= sqrt(R) 的全部素数，现筛区间 [L, R]
vector<char> bad(R - L + 1, 0);
if (L == 1) bad[0] = 1;                          // 极其重要：1 不是素数！
for (long long p : primes) {
    if (p * p > R) break;                        // 超过 sqrt(R) 的质数不需要再筛
    long long start = max(p * p, (L + p - 1) / p * p); // 防误删 p 本身
    for (long long j = start; j <= R; j += p)
        bad[j - L] = 1;                          // 偏移映射：j 对应下标 j - L
}
for (long long x = L; x <= R; ++x)
    if (!bad[x - L]) /* x 是区间内的素数 */;
```

本页要点

循环变量 p，start，j，x 全部必须用 long long，哪怕题目保证 $R < 2^{31}$！

为什么对？以及上机验证与反事实追问

- 为什么对：每个合数 $x \in [L, R]$ 必有质因子 $p \le \sqrt{x} \le \sqrt{R}$，且 $x \ge p^2$。因为 `start <= x <= R` 且每次加 p，x 必定会被这个 p 访问到并标记 `bad[x - L] = 1`。
- **3** 组上机验证（统计 $[L, R]$ 内素数个数）：
  1. `1 3` $\to$ 2 个（$2, 3$，检验 $L = 1$ 特判与不自删 $2, 3$）；
  2. `14 17` $\to$ 1 个（仅 17）；
  3. `2147483640 2147483647` $\to$ 1 个（仅 2147483647，检验 $2^{31} - 1$ 不死循环）。

反事实追问

把最后循环里的 `long long x` 改成 `int x`，输入 `2147483640 2147483647` 会发生什么？

当 $x = 2147483647$（即 `INT_MAX`）执行 `++x` 时，会瞬间溢出成 -2147483648，依然满足 `x <= R`，然后 `bad[x - L]` 负数下标越界 **RE / 死循环**！

本页要点

逼近 $2^{31} - 1$ 的闭区间遍历 `x <= R`，使用 `int` 必爆，只有 `long long` 能安全走完！

区间筛法常见错误对照表

错误写法	为什么错	正确写法与评测结果
漏写 `if (L == 1) bad[0] = 1;`	1 不会被任何质数倍数划掉	必须特判把 1 标记为非素数（避免 **WA**）
`start = (L + p - 1) / p * p`	当 $L \le p$ 时会把素数 p 自己划掉	写成 `max(p * p, ...)`（避免 **WA**）
`int x <= R; ++x`	$R = 2^{31} - 1$ 时 `++x` 溢出变负	全程用 `long long`（避免 **RE/TLE**）

本页要点

区间筛的三大坑口诀：“特判 1、max 防自杀、变量全开 long long”。

例题 1　先读懂要求：质因数分解（一本通 1620 / NOIP 2012）[入门]

大白话题意

已知正整数 n 恰好是两个不同的质数的乘积（$n = p_1 \times p_2$），请你求出其中较大的那个质数。

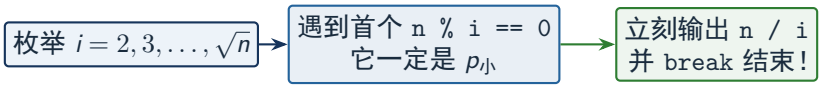
项目	具体要求
输入	一行 1 个正整数 n（$6 \le n \le 2 \times 10^9$）
处理	找出构成 n 的两个不同质数中较大的那一个
输出	一行 1 个正整数 p，即较大的那个质数
原题样例	输入 $21 \rightarrow$ 输出 7（因为 $21 = 3 \times 7$）

本页要点

题目直接保证了 n 只有两个质因子！不需要完整分解所有指数，只要找到最小质因子 i，答案就是 n/i！

例题 1　怎么想到的：找到最小的因子，大的那个就出来了

- 暴力思路：从 $n-1$ 往下找最大因子（需 $O(n)$ 步，$n=2\times10^9$ 时直接 **TLE**）。
- 利用性质：因为 $n=p_{小}\times p_{大}$，所以一定有 $p_{小}<\sqrt{n}<p_{大}$！
- 从 $i=2$ 开始从小到大枚举，第一个能整除 n 的整数 i 必定是较小质数 $p_{小}$（不可能是合数，否则它还有更小的因子！），于是 n/i 直接就是较大的质数 $p_{大}$！



本页要点

从小到大找第一个约数只需最多 $\sqrt{2\times10^9}\approx44721$ 步，远快于从大到小找！

例题 1　样例手推跟踪表

分别手推原题样例 `n = 21` 与小样例 `n = 15`：

输入 n	当前 i	`n % i == 0`?	动作	结论
21	2	$21 \bmod 2 = 1 \neq 0$	继续 `i++`	—
21	3	$21 \bmod 3 = 0$（真）	输出 $21/3 = \mathbf{7}$ 并退出	输出 7
15	2	$15 \bmod 2 = 1 \neq 0$	继续 `i++`	—
15	3	$15 \bmod 3 = 0$（真）	输出 $15/3 = \mathbf{5}$ 并退出	输出 5

本页要点

找到第一个能整除 n 的 i 后必须立刻 `break`（或 `return 0`），否则后面遇到 n/i 会再输出一次！

例题 1　核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 读入 n	$6 \le n \le 2 \times 10^9$	`i <= n / i` 会溢出吗？
2. 从 $i = 2$ 递增试除	第一个满足 `n % i == 0` 的 i	i 一定是质数吗？（是，最小非 1 约数必为质数）
3. 输出 n/i 并结束	较大的那个质数	加上 `break` 防止重复输出了吗？

相对质因数分解模板改了哪里？

因为题目保证 n 仅由两个不同质数相乘，不需要 `while (n % i == 0)` 除尽，也不需要末尾 `if (n > 1)`，找到第一个因子直接输出 `n / i` 即可！

本页要点

审清题目保证的条件，往往能把通用模板化简成极短的几行代码！

例题 1　核心代码（完整可运行程序）

```cpp
#include <iostream>
using namespace std;
int main() {
    long long n;
    cin >> n;
    for (long long i = 2; i <= n / i; ++i) { // 安全上限防溢出
        if (n % i == 0) {
            cout << n / i << '\n';          // i 是最小质因子，n/i 必为最大质因子
            break;                          // 找到后立刻退出
        }
    }
    return 0;
}
```

本页要点

用 `long long` 和 `i <= n / i`，哪怕 $n = 2 \times 10^9$ 也能稳稳在 1 毫秒内 AC！

例题 1　易错点与 3 组自测边界

- 易错 **1**：输出成较小的质数 `i`，题目问的是较大的那个质数 `n / i`！
- 易错 **2**：把 `i` 开成 `int` 且写 `i * i <= n`，若 n 接近 2×10^9，遇到较大 i 时平方溢出！

自测类型	输入数据	预期输出与验证要点
最小边界	6	输出 3（$6 = 2 \times 3$，检验 $i = 2$）
原题样例	21	输出 7（$21 = 3 \times 7$）
大数边界	2000000014（$= 2 \times 1000000007$）	输出 1000000007

本页要点

做“已知两质数之积”的题，交题前用 $n = 6$（最小合数）和 $n = 2 \times 1000000007$ 各测一次！

例题 2　先读懂要求：Prime Distance（一本通 1619）[提高]

大白话题意

多组询问，每次给 L, R（$1 \le L < R < 2^{31},\ R - L \le 10^6$），求 $[L, R]$ 中距离最近的相邻素数对 (C_1, C_2) 和最远的相邻素数对 (D_1, D_2)。并列取靠前一对；不足 2 个素数输出 `There are no adjacent primes.`。

项目	具体要求
输入	多组数据（直到文件尾 EOF），每行两个整数 L, R
处理	找出 $[L, R]$ 全部素数并求相邻差值最小与最大数对
输出	按格式输出 `C1,C2 are closest, D1,D2 are most distant.` 或无解提示语
原题样例	`2 17` → `2,3 are closest, 7,11 are most distant.`；`14 17` → 无解

本页要点

多组询问 + 严格输出格式（逗号、空格、句号），拼写错一个字符就会 PE 或 WA！

例题 2　怎么想到的：区间筛 + 双指针扫描相邻素数

- 第一步：因为 $R < 2^{31}$，在读入询问前先一次性筛出 $[2, 46341]$ 的所有小质数 `primes`（$46341^2 > 2^{31}$）。
- 第二步：对每组 $[L, R]$ 套用区间筛，得到标记数组 `bad[0..R-L]`。
- 第三步：从 $x = L$ 到 R 扫一遍，用 `prev` 记录上一个素数，遇到新素数 x 就更新最小差 `mn` 和最大差 `mx`（注意只在严格小于/大于时更新，就能自动保留靠前的一对！）。



本页要点

“存在多个并列时输出靠前的”，只需用严格不等号 < 和 > 更新即可，不需要额外特判！

例题 2　样例手推跟踪表（输入 2 17）

区间 $[2,17]$ 筛出的素数依次为 $2,3,5,7,11,13,17$，初始 `prev = -1`，`mn = INF`，`mx = -1`：

当前素数 x	上一个 prev	差值 $x-\text{prev}$	最近对 (c_1,c_2)（mn）	最远对 (d_1,d_2)（mx）
2	-1	—	未更新	未更新（prev 设为 2）
3	2	1	$(2,3)$，mn $=1$	$(2,3)$，mx $=1$
5	3	2	$2 \not< 1$，保留 $(2,3)$	$2>1$，更新 $(3,5)$，mx $=2$
7	5	2	$2 \not< 1$，保留 $(2,3)$	$2 \not> 2$，保留 $(3,5)$（靠前）
11	7	4	保留 $(2,3)$	$4>2$，更新 $(\mathbf{7},\mathbf{11})$，mx $=4$
13, 17	11, 13	2, 4	均不更小，保留 $(\mathbf{2},\mathbf{3})$	$4 \not> 4$，保留 $(\mathbf{7},\mathbf{11})$

本页要点

看 13 和 17 的差值也是 4，但因为用严格 >，所以完美保留了更靠前的 $(7,11)$！

例题 2　核心步骤表与相对模板的变化

按顺序计算	纸上得到	核对问题
1. 预处理 $B = 46341$ 内质数	`primes` 数组（约 4792 个）	只在 `while` 外预处理一次吗？
2. 对每组 $[L, R]$ 区间筛	`bad[0..R-L]`	特判 `L == 1` 和 `max(p*p, ...)` 了吗？
3. 扫 $[L, R]$ 更新最近/最远	`c1, c2, d1, d2`	循环变量 `x` 用 `long long` 了吗？
4. 按 `mx == -1` 判无解	对应格式化输出	逗号、空格、句号与原题一字不差吗？

- 相对模板改动：用 `prev` 扫相邻素数同时维护最小/最大差，最后按 `mx == -1` 判无解输出。

本页要点

切勿在 `while (cin >> L >> R)` 里重复预处理 46341 的质数表，放到循环外只算一次！

例题 2　核心代码（1/3）：预处理小质数（code/prime_distance.cpp）

```
const int B = 46341;
vector<int> primes; vector<char> comp(B + 1, 0);
for (int i = 2; i <= B; ++i) {
    if (!comp[i]) {
        primes.push_back(i);
        for (long long j = 1LL * i * i; j <= B; j += i) comp[j] = 1;
    }
}
```

本页要点

只在循环外预处理一次；注意内层 `j = 1LL * i * i` 用了 `long long` 防溢出。

例题 2　核心代码（2/3）：区间筛标记（code/prime_distance.cpp）

```
long long L, R;
while (cin >> L >> R) {
    vector<char> bad(R - L + 1, 0);
    if (L == 1) bad[0] = 1;              // 1 不是质数，必须特判
    for (long long p : primes) {
        if (p * p > R) break;
        long long start = max(p * p, (L + p - 1) / p * p);
        for (long long j = start; j <= R; j += p) bad[j - L] = 1;
    }
```

本页要点

每组询问只需开大小为 $R-L+1 \le 10^6+1$ 的 bad 数组，空间和时间都极省。

例题 2　核心代码（3/3）：扫描与输出（code/prime_distance.cpp）

```
    long long prev = -1, c1 = 0, c2 = 0, d1 = 0, d2 = 0;
    long long mn = LLONG_MAX, mx = -1;
    for (long long x = L; x <= R; ++x) if (!bad[x - L]) {
        if (prev != -1) {
            if (x - prev < mn) { mn = x - prev; c1 = prev; c2 = x; }
            if (x - prev > mx) { mx = x - prev; d1 = prev; d2 = x; }
        }
        prev = x;
    }
    if (mx == -1) cout << "There are no adjacent primes.\n";
    else cout << c1 << "," << c2 << " are closest, "
              << d1 << "," << d2 << " are most distant.\n";
}
```

本页要点

若区间内素数不足 2 个，mx 保持初值 -1，正好用来判无解！

例题 2　易错点与 3 组自测边界

- 易错 **1**：输出字符串末尾漏掉句号 `.`，或把 `closest`，后面漏掉空格。
- 易错 **2**：区间 $[L, R]$ 逼近 $2^{31}-1$ 时，`x` 或 `j` 开成 `int` 导致溢出死循环！

自测类型	输入数据	预期输出与验证要点
最小边界	`1 3`	`2,3 are closest, 2,3 are most distant.`（不能把 1 算进素数！）
无解边界	`14 17`	`There are no adjacent primes.`（只有 17 一个素数）
极限边界	`2147483640 2147483647`	`There are no adjacent primes.`（不溢出、不超时）

本页要点

自测 `1 3` 和 `2147483640 2147483647` 两组数据，可以一次性排掉区间筛 90% 的常见 Bug！

例题 3　读懂要求与思路：验证哥德巴赫猜想 [基础]

大白话题意

给定一个大于 4 的偶数 n（如 $n \le 10^6$），请把它拆成两个奇素数之和 $n = p_1 + p_2$。如果有多组解，输出 p_1 最小（即 $p_2 - p_1$ 最大）的那一组！

- 怎么想到的：先用欧拉线性筛一次预处理出 $[2, 10^6]$ 的素数表与标记数组 `is_comp[]`。
- 然后从小到大枚举奇素数 p_1（$p_1 \le n/2$），只要用 `!is_comp[n - p1]` 在 $O(1)$ 时间内判断 $n - p_1$ 是不是素数即可！

本页要点

“先线性筛预处理，再 $O(1)$ 查表验证 $n - p_1$”，是所有素数和类题目的标准套路！

例题 3　样例手推跟踪表（输入 $n = 20$）

枚举第 k 个奇素数	p_1	$p_2 = 20 - p_1$	`!is_comp[p2]?`	动作
1	3	17	真（17 是素数）	立刻输出 `20 = 3 + 17` 并 `break`！

- 再试 $n = 14$：$p_1 = 3 \to p_2 = 11$（是素数，输出 $3 + 11$）；若试 $n = 38$：$p_1 = 3 \to 35$（合数跳过），$p_1 = 5 \to 33$（合数跳过），$p_1 = 7 \to 31$（素数，输出 $7 + 31$）。

本页要点

因为是从小到大枚举 p_1，找到的第一个合法解自动就是 p_1 最小（差值最大）的解！

例题 3　核心代码与易错边界

```
// 假设已用线性筛预处理出 primes 与 is_comp[0..N]
int n;
while (cin >> n && n != 0) {
    for (int p : primes) {
        if (p == 2) continue;                // 题目要求奇素数，跳过偶素数 2
        if (p > n / 2) break;
        if (!is_comp[n - p]) {
            cout << n << " = " << p << " + " << n - p << '\n';
            break;
        }
    }
}
```

- 易错：在 `while (cin >> n)` 内重复跑筛法会 TLE；若要求“奇素数”须跳过 `p == 2`（如 $n = 6$ 输出 $3 + 3$）。

本页要点

多组询问题，预处理筛法永远放在最外层只执行一次！

本章常见错误对照表

题号 / 场景	常见错误写法	正确做法与原因
1619 区间筛	从 $L = 1$ 开始没特判 `bad[0]`	会把 1 当成素数！必须写 `if (L == 1) bad[0] = 1`
1619 区间筛	`start = (L + p - 1) / p * p`	当 $L \le p$ 时会把素数 p 自身筛掉，必须和 `p*p` 取 `max`
1619 区间筛	`int x <= R; ++x`（$R = 2^{31} - 1$）	`++x` 溢出变成负数死循环，全程用 `long long`
欧拉线性筛	筛完 `i * p` 漏写 `i % p == 0` 退出	失去线性性质且重复赋值，必须立即 `break`
1620 质因数分解	循环后漏判 `if (n > 1)`	会丢掉最大的那个大于 $\sqrt{n}$ 的质因子

本页要点

“1 不是质数要特判，区间起点防自杀，逼近 2^{31} 全开 `long long`，分解末尾检查 `n > 1`”！

按评测结果快速定位 Bug（CE / WA / RE / TLE / MLE）

评测结果	本章最常见原因	第一时间检查哪里
WA	把 1 当素数、自删小质数、漏掉 n > 1、输出标点少句号	测 1 3、14 17，逐字核对提示语与 `max(p*p,...)`
RE	`bad[]` 开小了或 `int` 溢出变负数下标	确认 `bad` 长度为 `R - L + 1`，且循环变量全是 `long long`
TLE	多组询问每次都重新筛 46341，或 $R = 2^{31} - 1$ 死循环	把小质数预处理放到 `while` 外；检查 `x` 是否为 `long long`
MLE	直接对 10^9 开全局数组	超过 10^7 范围不能普通筛，改用试除或区间筛

本页要点

区间筛错在 $L = 1$ 或小质数必查 `bad[0]` 与 `max(p*p,...)`；莫名 TLE 首先查 2147483647 时的 `int` 溢出！

提交前检查清单（六必查）

1. 试除判素数或分解质因数时，循环条件是否写成了 `i <= n / i` 且特判了 $n < 2$ 与末尾 `n > 1`？
2. 欧拉线性筛内层循环是否在 `1LL * i * p > N` 时截断，且先赋值 `minp[i*p]=p` 再 `if (i % p == 0) break;`？
3. 区间筛处理 $[L, R]$ 时，是否特判了 `if (L == 1) bad[0] = 1;`？
4. 区间筛计算起点时是否用 `max(p * p, (L + p - 1) / p * p)` 避免把素数 p 自己划掉？
5. 遍历 $[L, R]$ 和累加倍数 `j += p` 的变量是否全部声明为 `long long`，防止 $2^{31} - 1$ 溢出？
6. 若区间内只有 0 或 1 个素数（如 $[1, 1]$ 或 $[14, 16]$），输出提示语的拼写与句号是否和题面逐字一致？

最后一步

自测两组极端区间 `1 3` 和 `2147483640 2147483647`，确认不死循环且不把 1 算进素数后再提交！

本章全部练习与完成标准

难度	题号与链接	核心考点	配套源码
[入门]	学校 OJ：1620 ｜质因数分解	试除法找最小质因子、`n / i`	试除法模板
[基础]	哥德巴赫猜想	埃氏筛 / 线性筛预处理素数表	线性筛模板
[提高]	学校 OJ：1619 ｜ Prime Distance	区间筛法、`L==1`、`long long`	`code/prime_distance.cpp`
[提高]	学校 OJ：1629 ｜聪明的燕姿	约数和定理、质因子 DFS 搜索	约数和公式

完成标准

能独立默写 $O(n)$ 欧拉线性筛并说明 `if (i % p == 0) break;` 的数学理由；能无误写出区间筛 AC《Prime Distance》。

课后安排与下一章预告

- 复习：手推一遍 $N = 15$ 的欧拉线性筛过程，写下 12 和 9 分别是在哪一步、被哪个质数筛掉的。
- 练习：把线性筛改造成“同时记录每个数最小质因子 `minp[x]`”，再尝试分解 10^5 个随机整数验证速度。
- 下一章预告：第 3 章《同余与扩展欧几里得》——如何解方程 $ax + by = c$，以及为什么求逆元不一定非要模数是质数！

本章一句话

一切正整数都能拆成质数的积木；线性筛让每个合数只由最小积木建出，区间筛把二十一亿难题缩成百万窗口。

谢谢！