“挑战分解质因数”命题报告

张景行

2021 年 10 月 1 日

1 题目

1.1 题面

一天，小 Δ 给你了一个张纸，上面写了一个很大的数字 n ，他想让你在一秒钟内对它分解质因数，你定睛一看，发现这个数的二进制有高达 1500 位，然后你上网搜索了一下，发现连 RSA-1024 都还没有被分解出来，怎么可能一秒分出来呢?

但是突然你发现他在纸的背面也写了一些东西，你又定睛一看发现写的是 $\varphi(n)$，也就是 $\leq n$ 的且与 n 互质的正整数个数。

这下你好像明白咋分解了，现在你打开了你的电脑，复制出了你的祖传高精度整数运算库，你能一秒钟把结果告诉小 Δ 吗?

1.2 输入格式

一行，两个二进制整数 $n, \varphi(n)$。

1.3 输出格式

如果 n 可以表示为 $n = \prod_{i=1}^{T} p_i$，其中 p_i 是质数，且 $\forall 1 \leq i < T, p_i \leq p_{i+1}$，那么首先输出一行，包含一个十进制非负整数 T，接下来 T 行，每行一个二进制整数，代表 p_i，可以证明这样的表示是唯一的。

1.4 数据范围与限制

时间限制 1s，空间限制 256MB。

对于所有数据，$1 \leq n < 2^{1500}$。

本题采用捆绑测试，子任务如下:

1. $n < 2^{32}$ (5 分)
2. $n < 2^{64}$ (15 分)
3. $n < 2^{300}$，$n = pq$，其中 p, q 为不同的质数 (10 分)
4. $n < 2^{300}$, $n = \prod_{i=1}^{T} p_i$, 其中 p_i 为两两不同的质数, 且 $p_i \equiv 3 \pmod 4$ (15 分)
5. $n < 2^{300}$，$n = \prod_{i=1}^{T} p_i$，其中 p_i 为两两不同的质数 (15 分)
6. $n < 2^{300}$ (25 分)
7. 无特殊限制 (15 分)

保证每个子任务不超过 15 个测试点，但子任务之间会设置所有可行的依赖关系。

1.5 样例

1.5.1 样例 1

输入	输出
11111101 11011100	2 1011 10111

1.5.2 样例 2

见下发文件。

1.6 高精度库说明

我们在下发文件中提供了题面中提到的“祖传高精度整数运算库”，文件名是 `bigint.cpp`，直接将其内容全部复制到你的代码文件的开头即可使用，需要 C++11 或以上编译。

注意该代码使用了非标准的特性，需要在 x86-64 架构的 CPU ，和一个兼容 GCC 内联汇编的编译器（GCC，Clang）上编译和运行。若你的电脑不支持请使用自定义测试或者在线 IDE。

该库提供了一个结构体 `bigint`，其中可以存储任何长度的非负整数，其内部存储方式是一个 `std::vector<unsigned long long> data`，其中第 i 个元素的从低到高第 j 位是该整数的从低到高第 $64i+j$ 位（所有位数均从 0 开始记），并且存储过程中保证 `data` 的最后一个元素非零。

这种简单的的存储方式意味着若有需要你可以很方便实现你自己的函数。

令 n, m 为整数的位数，也就是最高的 1 所在位 $+1$，$w = 64$ 为位宽。

并且提供了关于 `bigint` 的以下函数：

默认的赋值运算符和拷贝构造函数。

空构造函数 `bigint()` ，会初始化为 0。

从整型的构造函数 `bigint(unsigned long long x)`，复杂度 $O(1)$。

从字符串的构造函数 `bigint(const std::string &s)`，注意不能有前导零，复杂度 $O(n)$。

到字符串的显式转换函数 `explicit operator std::string()const`，复杂度 $O(n)$。

到布尔的显式转换函数 `explicit operator bool()const`，会返回是否非零，复杂度 $O(1)$。

到整型的显式转换函数 `explicit operator unsigned long long()const`，若超出则对 2^{64} 取模，复杂度 $O(1)$。

`std::size_t digit()const`，会返回 n，复杂度 $O(1)$。

`bool operator==(const bigint &a)const`

`bool operator!=(const bigint &a)const`

`bool operator<(const bigint &a)const`

`bool operator>(const bigint &a)const`

`bool operator<=(const bigint &a)const`

`bool operator>=(const bigint &a)const`，整数的比较运算符，在 n, m 相同时复杂度 $O(n/w)$，否则 $O(1)$。

`bigint &operator<<=(std::size_t n)`,左移运算符,复杂度 $O(n/w)$。

`bigint &operator>>=(std::size_t n)`,右移运算符,复杂度 $O(n/w)$。

`bigint &operator+=(const bigint &a)`,加法运算符,复杂度 $O((n+m)/w)$。

`bigint &operator-=(const bigint &a)`，减法运算符，注意若结果是负数则行为未定义，复杂度 $O(n/w)$。

`bigint &operator*=(const bigint &a)`,乘法运算符,复杂度 $O(nm/w^2)$。

`bigint &operator/=(const bigint &a)`，除法（整除）运算符，若除数是 0 行为未定义，复杂度 $O(n(n-m)/w)$。

`bigint &operator%=(const bigint &a)`,取模运算符,若模数是 0 行为未定义，复杂度 $O(n(n-m)/w)$。

`bigint operator<<(std::size_t n)`

`bigint operator>>(std::size_t n)`

`bigint &operator++()`

`bigint &operator--()`

`bigint operator++(int)`

`bigint operator--(int)`

`bigint operator+(const bigint &a)const`

`bigint operator-(const bigint &a)const`

`bigint operator*(const bigint &a)const`

`bigint operator/(const bigint &a)const`

`bigint operator%(const bigint &a)const`, 二元运算符与自增自减运算符, 与上述一元运算符实现和复杂度相同。

还有以下非成员函数:

`std::istream &operator>>(std::istream &st,bigint &a)`,流输入。

`std::ostream &operator<<(std::ostream &st,const bigint &a)`, 流输出。

2 题解

2.1 算法一

直接使用试除法，复杂度 $O(\sqrt{n})$。期望通过子任务 1，得分 5 分。

2.2 算法二

使用 Pollard-Rho 算法，那么一轮分解的复杂度是 $O(n^{1/4})$。注意自带的高精度库的效率可能不足，可以直接使用普通的 64 位整数存储。

由于这个算法是指数级的，所以可以证明，总复杂度也是 $O(n^{1/4})$，直观则是考虑最坏情况显然是每次分出去一个 3，这样，因为 $n^{1/4}+(n/3)^{1/4}+(n/9)^{1/4}+\cdots=O(n^{1/4})$，所以总复杂度还是 $n^{1/4}$，严谨证明可以考虑数学归纳法易得，不做赘述。

所以期望通过子任务 1,2，得分 20 分。

2.3 算法三

当 p,q 是两个不同的质数的时候，$\varphi(pq)=(p-1)(q-1)$，而问题就变成了给定 $pq,(p-1)(q-1)$，求 p,q，这只需要解一个二次方程，可以考虑二分开根，复杂度 $O(\log^3 n/w^2)$，可以轻松通过子任务 3，得分 10 分。

2.4 算法三点五

考虑若我们枚举 $<n^{1/3}$ 的质数进行试除，那么最后会剩下最多两个质数，此时可以直接算出它们的 φ 值，就转化为了两个质因子的情况，这样就可以在 $O(n^{1/3})$ 的时间内分解，注意需要特判只有一个质数和是某个质数的平方的情况，与算法三结合期望通过子任务 1,2,3，得分 30 分。

2.5 算法四

考虑给定 φ 的作用，首先我们知道，任意数 a，若 $\gcd(a,n)=1$，则 $a^{\varphi(n)}=1$。但这个事实对分解质因数没有用。

不妨先考虑子任务 4 的限制，即 $n=\prod p_i$，且 $p_i=4k+3$。那么有 $\varphi(p_i)=4k+2=2t_i$（t_i 为奇数）。而测试点个数的保证则暗示着随机化算法。

考虑随机一个数 $1 < a < n$，首先若 $\gcd(a, n) > 1$，那么你直接就找到了一个非平凡因子。而 $\gcd(a, n) = 1$ 的时候，考虑 $a^{\prod t_i}$，发现，$a^{\prod t_j} \equiv 1 \pmod{p_i}$ 的概率是 $1/2$，且对于每一个 p_i 是独立的，这是因为考虑 a 的阶，其有 $1/2$ 的概率是偶数，有 $1/2$ 的概率是奇数。考虑如果我们找到了一个数 a，$a^{\prod t_j}$ 模某些 p_i 是 1，模另外一些不是，那么考虑 $\gcd(a-1, n)$，它其实就是一个非平凡因子，因为 $a-1$ 模某些 p_i 是 0，模另外一些不是。现在分析概率，当且仅当所有的都是 1 或都不是 1 时会失败，也就是令质因子个数为 T，概率是 $2/2^T$，若 $T \geq 2$，则失败概率 $\leq 1/2$。

注意上述论证不仅仅对 $\prod t_j$ 有效，也对其所有奇数倍的数有效，所以令 $\varphi(n) = t2^k$，其中 t 是奇数，那么选取 t 作为幂次则可以分解所有 n 的因数。

那么算法就明显了，我们递归地分解 n，每次选取一个合适的尝试次数 $S \approx 25$，然后随机选取 a，计算 $\gcd(a^t - 1, n)$，若答案不是 1 或 n 则递归下去。这样一轮的复杂度是 $O(S \log^3 n/w)$，瓶颈在快速幂。

而考虑分解的结构，由于实际上只有递归到质数才会跑满那 S 轮，其它的时候期望跑 $O(1)$ 轮就结束了（其实这个东西很像对于每个数跑一次 Miller-Rabin 素性测试），然后考虑递归的结构，发现由于每次每个质因子都会随机分配到两个儿子里面，所以期望递归 log 质因子个数轮，所以总复杂度为 $O((S + \log\log n)\log^3 n/w)$

期望通过子任务 4，得分 15 分。

2.6 算法五

考虑这个算法如何分解 p_i 不是 $4k+3$ 的情况。首先，先把 2 这个质因子给分掉，然后，考虑沿用之前的算法会出现什么问题，也就是沿用令 $\varphi(n) = t2^k$，其中 t 是奇数，那么选取 t 作为幂次分解的方式。

发现，由于某些 $\varphi(p_i) = t_i 2^{c_i}$，由于 c_i 不一定是 2 有可能很大，所以就会出现 $a^t \equiv 1 \pmod{p_i}$ 的概率不再是 $1/2$ 而有可能很小，就有可能分不出来。

但是考虑如果对于 $a^t = A$，考虑序列 $A, A^2, A^4, A^8, \cdots$，只要所有质因子不在同一项变成 1 就可以分解，即考虑生成 $A, A^2, A^4, A^8, \cdots$，直到 $A^{2^k} = 1$，然后考察 $\gcd(A^{2^{k-1}} - 1, n)$ 是否是非平凡因子。

考虑正确率，对于 $\varphi(p_i) = t_i 2^{c_i}$，在 A^{2^k} 处变为 1 的概率是

$$p_{i,k}=\begin{cases}1/2^{c_i} & k=0\\ 1/2^{c_i-k+1} & 1\le k\le c_i\\ 0 & k>c_i\end{cases}$$

原因也很简单，任取一个原根然后看指标中 2 的幂次的概率易得。

失败概率就是所有的都在同一时刻变成 1 的概率，即 $\sum_k\prod_i p_{i,k}$，发现，$\sum_k p_{i,k}=1$，且 $p_{i,k}\le 1/2$，那么若有 ≥ 2 个质因子，那么失败概率 $=\sum_k p_{1,k}\prod_{i\ge 2}p_{i,k}\le\sum_k p_{1,k}\times 1/2=1/2$。所以直接沿用算法四，并且加入一个不断乘方的过程，考虑乘方次数不超过 $O(\log n)$，所以一次分解的复杂度不变。

但是注意此时质因子不再是平均分的，考虑若我们有若干个质因子，它们的 c_i 都差距很大，那么显然很大概率会是按 c_i 从大到小依次剥离每个质因子，所以有可能递归高达 $O(\log n)$ 而不是 $O(\log\log n)$ 层。所以总复杂度是 $O((S+\log n)\log^3 n/w)$，但依然可以通过。（实际上这 log 层很难操作使得每一部分的复杂度都卡满，所以表现差不多）

期望通过子任务 3,4,5，得分 45 分。

2.7 算法六

考虑算法五在此时会出什么问题，发现由于 $\bmod p_i^{k_i}$ 还有原根，所以其他结论依然不变，在有 ≥ 2 个质因子的情况下依然可以正常分解，那么问题出在只有一个质因子的情况下，也就是无法分解 p^k，这个时候，只需要在分解失败后，枚举 m，计算 $\lfloor\sqrt[m]{n}\rfloor^m$ 是否等于 n 即可。如果使用二分实现的话，考虑计算 $\lfloor\sqrt[m]{n}\rfloor$，需要二分 $O(\log n/m)$ 位，计算乘方的时候，考虑快速幂每次会长度加倍，而总长度是 $O(\log n)$ 的，所以乘方的复杂度是 $O(\log^2 n/w^2)$，所以总复杂度是 $O(\log^3 n/mw^2)$，对于所有 m 计算则是 $O(\log^3 n\log\log n/w^2)$，注意到只有递归到叶子才会进行这种分解，故这部分的总复杂度也不超过 $O(\log^3 n\log\log n/w^2)$。

但注意质数幂次带来的问题，有可能会有诸如 3^k 这样的数字，虽然理论上不能分解，但是由于是小质数，所以会在上一步被成功分解从而导致复杂度退化到 $O(S\log^4 n/w)$，可以对 <100 的质数来试除来规避这种问题。

期望通过子任务 1,2,3,4,5,6，得分 85 分。

2.8 算法七

考虑对算法六进行优化。

首先复杂度分为两部分，试开根的 $O(\log^3 n \log\log n/w^2)$，以及每次分解快速幂的 $O((S+\log n)\log^3 n/w)$。

注意每次分解的快速幂耗时 $O(\log^3 n/w)$，gcd 等只耗 $O(\log^2 n/w)$，而发现快速幂包括了乘法和取模，显然瓶颈在于取模，而我们知道 Barrett 模乘和 Montgomery 形式均可以把取模优化成乘法，所以可以将快速幂优化成 $O(\log^3 n/w^2)$。

直接实现后，复杂度是

$$O((S+\log n)(\log^3 n/w^2+\log^2 n/w)+\log^3 n\log\log n/w^2)$$

还可以进行其他优化：考虑对小质数进行试除，若我们对 $< B$ 的质数进行试除，那么复杂度是 $O\left(B+\frac{B\log n}{w\log B}\right)$，而这会使分解的素因子个数 $/\log B$。这个优化的具体最优复杂度，以及对应 B 的取值不好分析，可以选取一个尽可能大的 B，能使效率得到一定的提升。

进行上述优化后，期望通过所有测试点，得分 100 分。

3 参考资料

算法来源：

A Computational Introduction to Number Theory and Algebra，10.4 Factoring and computing Euler’s phi function，https://shoup.net/ntb/ntb-v2.pdf