Метод квадратичного решета разложения целых чисел на множители

Алгоритм

Для начала введем несколько теорем и определений

1. Будем называть число m B-гладким, если разложение его на простые множители не содержит чисел больших B
2. Если существуют a и b такие, что $a^2 - N = b^2$ и $a \neq \pm b \mod N$, тогда $\gcd(a \pm b, N)$ это его разложение
3. Если N имеет как минимум два различных нечетных простых делителя, тогда как минимум половина решений $a^2 = b^2 \mod N$ с $\gcd(a * b, N) = 1$ удовлетворяет (2)
4. Если $m_1, m_2, ..., m_k$ положительные B-гладкие числа, и $k > \pi(B)$, тогда произведение некоторой непустой подпоследовательности $[m_i]$ является полным квадратом
5. Символ Лежандра от числа a и простого числа p
   - 0, если a делится на p
   - 1, если a является квадратичным вычетом по модулю p (то есть существует такое целое x, что $x^2 \equiv a \mod p$) и a не делится на p
   - -1, если a является квадратичным невычетом по модулю p (иначе говоря все остальные случаи)
6. Алгоритм Тонелли-Шенкса используется для решения сравнения $x^2 \equiv n \mod p$, где n — квадратичный вычет по модулю p, а p — нечётное простое число
   Результат работы алгоритма: вычет x, удовлетворяющий сравнению $x^2 \equiv n \pmod p$.
   (a) Выделим степени двойки из $p - 1$, то есть пусть $p - 1 = 2^S Q$, где Q нечётно, $S \geq 1$. Заметим, что если $S = 1$, то есть $p \equiv 3 \pmod 4$, тогда решение определяется формулой $x \equiv \pm n^{\frac{p+1}{4}}$. Далее полагаем $S = 2$.
   (b) Выберем произвольный квадратичный невычет z, то есть символ Лежандра $\left(\frac{z}{p}\right) = -1$, положим $c \equiv z^Q \pmod p$
   (c) Пусть также $x \equiv n^{\frac{Q+1}{2}} \pmod p, t \equiv n^Q \pmod p, M = S$.
   (d) Выполняем цикл:
      i. Если $t \equiv 1 \pmod p$, то алгоритм возвращает x.
      ii. В противном случае в цикле находим наименьшее i, $0 < i < M$, такое, что $t^{2^i} \equiv 1 \pmod p$ с помощью итерирования возведения в квадрат.
      iii. Пусть $b \equiv c^{2^{M-i-1}} \pmod p$, и положим $x :\equiv xb \pmod p$, $t :\equiv tb^2 \pmod p$, $c :\equiv b^2 \pmod p$, $M := i$, возвращаемся к началу цикла.
   (e) После нахождения решения сравнения x второе решение сравнения находится как $p - x$.

Данные теоремы накладывают несколько ограничений на N

- Число составное
- Число имеет как минимум два различных нечетных простых делителя

- Число не имеет простых делителей до десятичного логарифма (при наличии таких делителей алгоритм работает менее эффективно)

Алгоритм факторизации N

- Посчитать параметр B для числа
- Рекомендуемая оценка на $B = e^{0.5 \log N \log \log N}$
- С помощью символа Лежандра отберем простые числа для разложения, простое число нам подходит, если N является квадратичным вычетом по его модулю, тоесть символ Лежандра равен единице. Назовем эти простые числа базовыми
- Для ускорения просеивания будем использовать алгоритм Тонелли - Шенкса, он помогает сократить пространство поиска кандидатов для просеивания
- Посчитать $x^2 - N$ и проверить, что оно B-гладкое для $x \geq \sqrt{N}$, повторять это действие пока не наберется $\pi(B)$ чисел
- Сформировать матрицу M со столбцами, состоящими из экспоненциального представления в виде вектора каждого гладкого числа. Найти линейную зависимость, вычислив ядро этой матрицы
- Построить a, b способом, соответствующим теореме 2. Если $a \equiv b \mod N$, то начать с шага 1 (таких повторений не будет слишком много по теореме 3)
- Для ускорения первого шага планируется воспользоваться теоремой 4

Архитектура программы

- Для длинной арифметики используется **GMP**
- Алгоритм находится в классе **Quadratic_Sieve** программы
- Разложение на множители производится с помощью метода **factorize** этого класса
- В первую очередь отберем подходящие простые числа в отрезке $[2, B]$ с помощью символа Лежандра, простое число нам подходит, если N является квадратичным вычетом по его модулю, тоесть символ Лежандра равен единице
- Далее запустим алгоритм просеивания, который запускается поочередно на нескольких отрезках (длину отрезка можно регулировать во входных данных). Пусть мы просеиваем отрезок $[a, b]$, для каждого базового p первое кратное число в отрезке это $\lceil \frac{a}{p} \rceil p$, найдем так же $s^2 \equiv N \mod p$ с помощью алгоритма Тонелли-Шенкса, описанного в разделе 6. Тогда вероятными кандидатами будут x из $\{\lceil \frac{a}{p} \rceil p + ip - s, \lceil \frac{a}{p} \rceil p + ip + s\}$ при $i \in [0..]$, если x подходит, пробуем разложить $x^2 - N$ по базовым простым, и если получается переходим к следующему числу
- Во время разложения числа на множители необходимо хранение B-гладких чисел, оно осуществляется в виде хранения самих чисел, так и в виде разреженного вектора, содержащего номера простых чисел, входящих в исходное в нечетной степени (это ровно то пространство векторов, ядро которого необходимо для нахождения ответа)
- После нахождения необходимого количества B-гладких чисел, их матричное представление загружается в класс **NumberRepresentation** в виде битового представления внутри 64-битных чисел. После чего **get_sparce_nullspace** приводит данную матрицу к ступенчатому виду. Для этого необходимо отсортировать строчки матрицы по наиболее раннему вхождению единицы. После этого поочередно прибавлять i-тую строку ко всем последующим, в которых есть первая единица на том же месте. Данный алгоритм имеет асимптотику $\mathcal{O}(n^3)$, однако за счет использования специфического представления скорость работы сильно увеличивается
- Нахождение базиса ядра заменено на выбор случайного вектора из ядра, из этого следуют минусы случайных алгоритмов, однако сравнивая производительность этого варианта и варианта из предыдущего решения виден выйгрыш в несколько раз. Полученный вектор необходимо сконвертировать в номера единиц, и все соответствующие простые числа перемножить. Мы получили число a, и можем попробовать получить разложение по формуле из Теоремы 2, но в случае неудачного разложения необходимо повторить попытку с другим вектором
- Типом возвращаемого значения этой функции и так же функции **factorize** является экземпляр класса **GMP**

Тестирование

- В данном проекте применялось модульное тестирование, соответственно покрытие тестами функции **factorize** можно найти в коде функции **main**.
- Для тестирования функции **factorize** было создано множество простых тестов с числами различных размерностей и с различным количеством делителей, под каждый из которых было подобрано оптимальное B. Так же во время тестирования производительности (конечно же вне замера времени) происходила валидация результата.

Производительность

Производительность замерялась на двухядерном процессоре (четыре виртуальных ядра) intel i5, поэтому размер чисел был сильно ограничен для полного покрытия промежутка. При этом на графиках хорошо видна низкая скорость роста затраченного времени при $B = 300 + \exp^{0.5 \log\log n \log n}$. Асимптотика алгоритма демонстрирует низкую скорость роста. В результатах виден некоторый шум, он вызван специфическим алгоритмом перебора векторов в ядре (замер времени усреднялся по 300 попыткам разложения), а так же падением эффективности алгоритма на числах с маленькими делителями.

Рисунок 1. Зависимость скорости работы от числа в диапазоне 10^5 до 10^7.