Поиск дискретного логарифма

Сергей Николенко

Криптография — АУ РАН, осень 2011

Outline

Промежуточный итог

- Итак, по итогам первых двух фаз мы вычислили $\log_g p_i$ для $p_i \leq B$. Как теперь найти $\log_g y$?
- Мы будем брать случайные числа w, пока yg^w не станет достаточно гладким.
- Но здесь «достаточно» не B-гладкости, а U-гладкости для некоторого $U > B$ (все константы выберем потом, когда будем сложность оценивать).

Идея третьей фазы

- Итак, выбираем w и проверяем yg^w на U-гладкость (заодно раскладывая на множители).
- Затем, когда yg^w станет U-гладким, задача сведётся к логарифмированию нескольких простых чисел «среднего размера» (от B до U). Такое простое m мы логарифмируем так.
  1. Начиная с $u = \lceil \sqrt{p}/m \rceil$ и увеличивая u, найдём B-гладкое u.
  2. Начиная с $v = H = \lceil \sqrt{p} \rceil$ и увеличивая v, найдём B-гладкое
  $$n \equiv uvm \pmod{p}.$$
  3. Теперь $\log_g m = \log_g n - \log_g u - \log_g v$, и все логарифмы справа мы знаем.
- Оба числа u и v можно найти полиномиальным решетом (оба многочлена линейные).

О равномерной сложности дискретного логарифма

- Обратите внимание: все дискретные логарифмы искать одинаково трудно.
- Если какой-нибудь $\log_g y$ было бы труднее вычислить, чем для большинства других y, достаточно было бы брать случайные w, пока yg^w не стало бы легко логарифмировать.
- А логарифмы по другому основанию, если умеем искать логарифмы по основанию g, тоже искать несложно, ведь

$$\log_h a \equiv \frac{\log_g a}{\log_g h} \pmod{p-1}.$$

Какие есть параметры

- Итак, мы хотим найти оптимальные параметры для алгоритма index calculus.
- Параметры — это:
  - B — базовая оценка гладкости;
  - C — число, до которого варьируются $0 \leq c_1 < c_2 \leq C$ в решете;
  - U — новая оценка гладкости на последнем этапе.
- Для начала предположим, что третья фаза быстрее первых двух, и соптимизируем B и C.

Числа $L_p[s; c]$

- Вспомним обозначения:

$$L_p[s; c] = e^{c(\log n)^s(\log\log n)^{1-s}}.$$

- Мы сейчас всё будем делать в терминах $L_p[s; c]$, поэтому сначала установим простые свойства $L_p[s; c]$.
- Замечание: мы будем включать все константные множители внутрь L_p, т.е. читать L_p как $O(\ldots)$.

Числа $L_p[s; c]$

- Крайние случаи:

$$\text{если } s = 0, L_p[s; c] = (\log p)^c \quad \text{(полиномиальная сложность);}$$
$$\text{если } s = 1, L_p[s; c] = e^{c \log p} \quad \text{(экспоненциальная сложность).}$$

- Сумма:

$$L_p[s_1; c_1] + L_p[s_2; c_2] = L_p[\max\{s_1, s_2\}; \max\{c_1, c_2\} + o(1)]$$

(на самом деле $\max\{c_1, c_2\}$ — это только для случая $s_1 = s_2$, но в любом случае это верхняя оценка, и нам её хватит).

- Произведение:

$$L_p[s_1; c_1] \cdot L_p[s_2; c_2] = L_p[\max\{s_1, s_2\}; c_1 + c_2 + o(1)]$$

(то же замечание про $c_1 + c_2$).

Количество гладких чисел

- Итак, будем оптимизировать B и C.
- Сначала повторим и расширим некоторые рассуждения из прошлой лекции.
- Теорема из теории чисел (без доказательства): для любого $\epsilon > 0$, если $X \to \infty$, $u \to \infty$, причём $X^{1/u} > (\log X)^{1+\epsilon}$, то

$$\frac{\psi(X, X^{1/u})}{X} = u^{-(1+o(1))u},$$

где $\psi(X, B)$ — количество B-гладких чисел от 1 до X.
- Если $B = X^{1/u}$, значит, $u = \frac{\log X}{\log B}$.

Количество гладких чисел

- Нас интересуют B и X вида $L_p[s;c]$; подставим $X = L_p[s;c]$ и $B = L_p[s_B;c_B]$ в эту формулу:

$$\frac{\psi(X,B)}{X} = u^{-(1+o(1))u} =$$

$$= \left(\frac{c(\log p)^s(\log\log p)^{1-s}}{c_B(\log p)^s_B(\log\log p)^{1-s_B}}\right)^{-\frac{c(\log p)^s(\log\log p)^{1-s}}{c_B(\log p)^s_B(\log\log p)^{1-s_B}}+o(u)} =$$

$$e^{(s-s_B)\frac{c}{c_B}(\log p)^{s-s_B}(\log\log p)^{-s+s_B}(\log\log p+O(\log\log\log p))} =$$

$$L_p\left[s-s_B; -(s-s_B)\frac{c}{c_B}+o(1)\right].$$

- Это вероятность того, что случайное число от 1 до X будет B-гладким. Как обычно, про значения многочленов мы ничего не знаем, только предполагаем.

Количество гладких чисел

- Всего в нашей базе факторизации $\pi(B) \approx \frac{B}{\log B}$ простых чисел.
- Итого, если нам нужны $\frac{DB}{\log B}$ соотношений, а гладким будем каждое u^u число, мы должны выполнить
$$\frac{DBu^u}{\log B}$$
тестов на гладкость.
- Здесь мы, конечно, воспользуемся решетом и получим, что общее время на генерацию системы соотношений равно
$$\frac{DBu^u}{\log B} \log\log B.$$
- Найдём минимум этого значения по B.

Оптимизация

- Перейдём к логарифму: минимизируем теперь

$$\log D + \log B + u \log u - \log\log B + \log\log\log B.$$

- Возьмём производную по B и приравняем нулю:

$$\frac{1}{B} + \frac{du}{dB}\log u + \frac{du}{dB} = 0.$$

- Вспомним, что $u = \frac{\log X}{\log B}$:

$$\frac{1}{B} - \frac{\log X \log u}{B(\log B)^2} - \frac{\log X}{B(\log B)^2} = 0,$$
$$\log X(1 + \log\log X - \log\log B) = (\log B)^2.$$

Оценка

- Мы получили, что

$$\log X(1+\log\log X-\log\log B)=(\log B)^2.$$

- Поскольку $1<\log\log B<\log\log X$,

$$\log X<\log X(1+\log\log X-\log\log B)<\log X\log\log X, \text{ и}$$

$$e^{\sqrt{\log X}}<B<e^{\sqrt{\log X\log\log X}}.$$

- Раз уж мы ищем B в виде $L_p[s_B;c_B]$, это значит, что оптимальный выбор — что-то в духе

$$B=L_p\left[\frac{1}{2};c_B\right]$$

для некоторого c_B.

Сколько же на самом деле проверок

- Мы там ничего не говорили о D; а оно связано с C и, в конечном счёте, B.
- Поэтому сейчас оценим поточнее. Пусть $B = L_p[s_B; c_B + o(1)]$, $C = L_p[s_C; c_C + o(1)]$; напоминаю, что C — это оценка на c_1 и c_2.
- Мы проверяем все $0 \leq c_1 < c_2 \leq C$, то есть всего будет проверок
$$\frac{1}{2}C^2 = L_p[s_C; 2c_C + o(1)].$$
- А всего гладких чисел нужно найти
$$B + C = L_p[s_B; c_B + o(1)] + L_p[s_C; c_C + o(1)] =$$
$$= L_p[\max\{s_B, s_C\}; \max\{c_B, c_B\} + o(1)].$$

Вывод точной оценки

- Если P_{sm} — вероятность обнаружить гладкое число, то нужно выбрать B и C так, чтобы

$$\frac{1}{2}C^2 P_{sm} \geq B + C.$$

- Какого порядка будут эти числа? Мы брали числа вида $x = (H + c_1)(H + c_2)$, где $H = \lceil \sqrt{p} \rceil = \lceil L_p \left[1; \frac{1}{2}\right] \rceil$. Поскольку $J = H^2 - p \leq 2H$:

$$x = J + (c_1 + c_2)H + c_1 c_2 \leq (2 + c_1 + c_2)H + c_1 c_2 \leq$$

$$\leq 2L_p[s_C; c_C + o(1)] L_p\left[1; \frac{1}{2}\right] + L_p[s_C; 2c_C + o(1)] = L_p\left[1; \frac{1}{2} + o(1)\right].$$

Вывод точной оценки

- А вероятность P_{sm}, как мы уже говорили,
$$P_{sm} = \frac{\psi(x, B)}{x} = L_p\left[1 - s_B; \frac{-(1-s_B)}{2c_B} + o(1)\right].$$
- Тогда условие $\frac{1}{2}C^2 P \geq B + C$ превращается в
$$L_p[s_C; 2c_C + o(1)] L_p\left[1 - s_B; \frac{-(1-s_B)}{2c_B} + o(1)\right] \geq$$
$$\geq L_p[\max\{s_B, s_C\}; \max\{c_B, c_B\} + o(1)],$$ то есть
$$L_p[s_C; 2c_C + o(1)] \geq$$
$$\geq L_p[\max\{s_B, s_C\}; \max\{c_B, c_C\} + o(1)] L_p\left[1 - s_B; \frac{(1-s_B)}{2c_B} + o(1)\right].$$
- Отсюда, как минимум (точнее позже), $s_C \geq \max\{s_B, s_C, 1 - s_B\}$.

Оптимизация

- С другой стороны, давайте вернёмся к времени работы.
- Решето наше C раз проверяет по C чисел (фиксирует c_1 и варьирует c_2), то есть работает время

$$C \cdot \left(\pi(B)(1+\log B)^{o(1)} + C \log\log B\right) =$$
$$= L_p[s_C; c_C]\left(L_p[s_B; c_B] + L_p[s_C; c_C]\right) =$$
$$= L_p[\max\{s_B, s_C\}; c_C + \max\{c_B, c_C\} + o(1)].$$

- А на линейную алгебру нужно время

$$(B+C)^2 = L_p[\max\{s_B, s_C\}; \max\{2c_B, 2c_C\} + o(1)].$$

Оптимизация

- В итоге первая и вторая фазы занимают

$$L_p[\max\{s_B, s_C\}; \max\{2c_B, 2c_C\} + o(1)].$$

- Нужно минимизировать в первую очередь $\max\{s_B, s_C\}$ при условии

$$s_C \geq \max\{s_B, 1 - s_B\}.$$

- Получается $s_B = s_C = \frac{1}{2}$. При этом

$$P_{sm} = L_p\left[1 - s_B; \frac{-(1 - s_B)}{2c_B} + o(1)\right] = L_p\left[\frac{1}{2}; -\frac{1}{4c_B} + o(1)\right].$$

Оптимизация

- Т.к. $P_{sm} = L_p\left[\frac{1}{2}; -\frac{1}{4c_B} + o(1)\right]$, условие на достаточное количество гладких чисел $\frac{1}{2}C^2P \geq B + C$ превращается в

$$L_p[\frac{1}{2}; 2c_C + o(1)] \geq L_p\left[\frac{1}{2}; \max\{c_B, c_C\} + o(1)\right] L_p\left[\frac{1}{2}; \frac{1}{4c_B} + o(1)\right],$$

$$\text{то есть } 2c_C \geq \max\{c_B, c_C\} + \frac{1}{4c_B}.$$

Оптимизация

- А суммарное время работы алгоритма превращается в

$$L_p[\max\{s_B, s_C\}; \max\{2c_B, 2c_C\} + o(1)].$$

- Оптимизируя это при условии $2c_C \geq \max\{c_B, c_C\} + \frac{1}{4c_B}$, получим $c_B = c_C = \frac{1}{2}$.
- В итоге $B = C = L_p\left[\frac{1}{2}; \frac{1}{2} + o(1)\right]$, а суммарное время работы первой и второй фаз составляет

$$L_p\left[\frac{1}{2}; 1 + o(1)\right].$$

Время работы третьей фазы алгоритма

- Мы предполагали, что третья фаза будет быстрее первых двух. Верно ли это?
- Напоминаю, что мы выбираем w и проверяем yg^w на U-гладкость, пока не попадём.
- Давайте оценим; у нас теперь новый параметр $U = L_p\left[s_U; c_U + o(1)\right]$, а вероятность найти подходящее число w будет P_w:

$$P_w = \frac{\psi(p, U)}{p} = L_p\left[1 - s_U; -\frac{1 - s_U}{c_U} + o(1)\right].$$

Время работы третьей фазы алгоритма

- Если мы пользуемся ECM, то каждое число проверяется за

$$e^{\sqrt{(2+o(1))\log U \log\log U}}(\log n)^2 =$$
$$= L_p\left[\frac{s_U}{2}; \sqrt{2s_U c_U} + o(1)\right].$$

- А нам нужно провести $\frac{1}{P_W}$ таких тестов, т.е. общее время на поиск w равно

$$L_p\left[\frac{s_U}{2}; \sqrt{2s_U c_U} + o(1)\right] \cdot L_p\left[1 - s_U; \frac{1 - s_U}{c_U} + o(1)\right] =$$
$$= L_p\left[\max\{\frac{s_U}{2}, 1 - s_U\}; \frac{1 - s_U}{c_U} + \sqrt{2s_U c_U} + o(1)\right].$$

Время работы третьей фазы алгоритма

- Итак, нужно оптимизировать

$$L_p\left[\max\{\frac{s_U}{2}, 1-s_U\}; \frac{1-s_U}{c_U} + \sqrt{2s_Uc_U} + o(1)\right].$$

- Минимизируя $\max\{\frac{s_U}{2}, 1-s_U\}$, получим $s_U = \frac{2}{3}$, а минимизируя $\frac{1}{3c_U} + 2\sqrt{c_U/3}$, получим $c_U = \left(\frac{1}{3}\right)^{1/3}$. Итак:

$$U = L_p\left[\frac{2}{3}; \left(\frac{1}{3}\right)^{1/3} + o(1)\right],$$

а общее время работы третьей фазы составляет

$$L_p\left[\frac{1}{3}; 3^{1/3} + o(1)\right].$$

Анализ

- У нас получилось $L_p\left[\frac{1}{3}; 3^{1/3} + o(1)\right]$, что гораздо быстрее, чем $L_p\left[\frac{1}{2}; 1 + o(1)\right]$ (время первой и второй фазы).
- Но так получилось только благодаря ECM; если использовать для проверки на гладкость метод Полларда, получится то же самое $L_p\left[\frac{1}{2}; 1 + o(1)\right]$, а с тривиальным алгоритмом проверки (пробным делением) и вовсе $L_p\left[\frac{1}{2}; \sqrt{2} + o(1)\right]$.

Упражнение. Доказать эти оценки.

Но это ещё не всё

- Нужно ещё оценить логарифмирование «среднего размера» простых чисел.
- Нам для каждого такого простого m надо найти B-гладкое $u > \sqrt{p}/m$. Здесь u — число порядка $L_p[1; \frac{1}{2}]$, а вероятность выбрать гладкое u — $L_p\left[\frac{1}{2}; -\frac{1}{2} + o(1)\right]$.
- Т.е. нужно прогнать через решето $L_p\left[\frac{1}{2}; \frac{1}{2} + o(1)\right]$ вариантов; это быстрее первой и второй фазы.
- А самый последний шаг — найти такое $v > \sqrt{p}$, что $uvm \pmod{p}$ будет B-гладким. Здесь v тоже порядка $L_p[1; \frac{1}{2}]$, и точно так же получается сложность $L_p\left[\frac{1}{2}; \frac{1}{2} + o(1)\right]$.
- Так что этот шаг оказался сложнее, чем «основная часть» третьей фазы, но всё равно быстрее первой и второй фазы.

Теперь всё

- Теперь всё. Уффф.
- Важное замечание: одни и те же результаты первой и второй фазы можно использовать для вычисления многих дискретных логарифмов; каждый новый логарифм будет стоить как третья фаза, а не как первая+вторая, что дешевле.

Outline

Решето числового поля

- Полиномиальное решето — не предел мечтаний.
- Ещё эффективнее оказывается метод *решета числового поля* (number field sieve).
- По сути метод аналогичен квадратичному решету, но теперь всё происходит над другими кольцами.
- Мы рассмотрим только основную идею, безо всяких доказательств.

Идея

- Мы рассмотрим решето числового поля для задачи разложения чисел на множители.
- Мы хотим разложить n. Предположим, что у нас есть неприводимый многочлен $f(x)$ и число m, такое, что $f(m) \equiv 0 \pmod{n}$.
- Рассмотрим комплексный корень α многочлена $f(x)$ и кольцо $\mathbb{Z}[\alpha]$.
- $f(m) \equiv 0 \pmod{n}$ и $f(\alpha) = 0$, следовательно, есть естественный гомоморфизм колец $\varphi : \mathbb{Z}[\alpha] \to \mathbb{Z}_n$, который отображает α в m.

Идея

- Теперь предположим, что у нас есть множество таких пар чисел (a, b), что:
  - произведение всех $(a - \alpha b)$ — квадрат в кольце $Z[\alpha]$, скажем, γ^2;
  - произведение всех $(a - mb)$ — квадрат в $\mathbb{Z}$, скажем, v^2.
- Заменим в выражении для γ α на m; получим $\varphi(\gamma) \equiv u \mod n$. Теперь

$$u^2 \equiv \varphi(\gamma)^2 = \varphi\left(\gamma^2\right) = \varphi\left(\prod (a - \alpha b)\right) =$$
$$= \prod \left(\varphi\left(a - \alpha b\right)\right) = \prod (a - mb) = v^2 \pmod{n},$$

и мы тем самым сможем разложить n на множители.

Многочлен f

- Но откуда взять f? Он сам собой появится.
  - Выберем степень d, положим $m = \lfloor n^{1/d} \rfloor$.
  - Запишем n по основанию m: $n = m^d + c_{d-1}m^{d-1} + \ldots + c_0$.
  - Вот и многочлен: $f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_0$.
- Отдельный вопрос: будет ли он неприводимым? Если не будет, то $n = f(m) = g(m)h(m)$, и мы уже (с высокой вероятностью) разложили n. Так что будет. :)

Числа a и b

- Откуда взять a и b? Из такого же решета.
- Чтобы $\prod(a - mb)$ было квадратом, нужно решить линейную систему на коэффициенты, как раньше.
- Чтобы $\prod(a - \alpha b)$ было квадратом, нужно решить линейную систему на коэффициенты в кольце $\mathbb{Z}[\alpha]$, если это хорошее кольцо (с единственностью разложения). Хорошее кольцо можно добыть (без д-ва).
- Теперь можно просто объединить две системы — нам нужно, чтобы оба свойства выполнялись.

Оценка сложности

- Чем хорошо решето числового поля?
- Наши оценки были основаны на X — количестве чисел, из которых можно сделать квадрат.
- У нас было $X = n^{1/2+\epsilon}$.
- А в number field sieve получается $X = e^{c(\log n)^{2/3}(\log\log n)^{1/3}}$, что даёт общую оценку сложности
$$L_n\left[\frac{1}{3}; c\right] = e^{(c+o(1))(\log n)^{1/3}(\log\log n)^{2/3}}.$$
- Теоретический рекорд: $c \approx 1,902$, из анализа нашего алгоритма получилось бы
$$L_p\left[\frac{1}{3}; \left(\frac{64}{9}\right)^{1/3} + o(1)\right] \approx L_p\left[\frac{1}{3}; 1,923 + o(1)\right].$$
- Но главное — основная асимптотика стала лучше.

Number field sieve для дискретного логарифма

- Аналогичные соображения проходят и для задачи дискретного логарифма, и тоже время работы получается

$$L_p\left[\frac{1}{3};\left(\frac{64}{9}\right)^{1/3}+o(1)\right]\approx L_p\left[\frac{1}{3};1,923+o(1)\right].$$

- На практике решето числового поля начинает выигрывать, где-то начиная со 100-значных чисел.

Решётки

- Мы уже говорили, насколько важны в криптографии решётки, а точнее, задача поиска кратчайшего вектора.
- Оказывается, с их помощью можно и числа раскладывать!
- Schnorr, 1993; Adleman, 1995; Schnorr, 2008.
- Пока (кажется) это недостаточно практичный способ, хуже number field sieve, но кто знает, к чему придёт...
- Мы вкратце опишем метод, без подробных доказательств и оценок сложности.

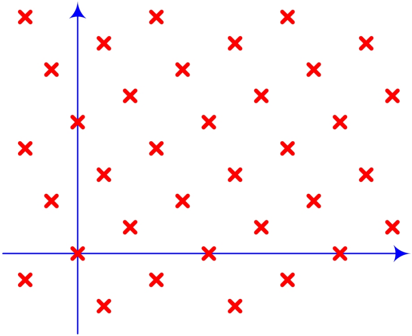
Решётки

- Будем обозначать скалярное произведение как $\langle x, y \rangle$.
- Пусть $B = \{b_1, \ldots, b_m\}$ — набор линейно независимых векторов в $\mathbb{R}^n$. Тогда *решётка* размерности m — это набор линейных комбинаций b_i с целыми коэффициентами:

$$L = \mathbb{Z}b_1 + \mathbb{Z}b_2 + \ldots + \mathbb{Z}b_m.$$

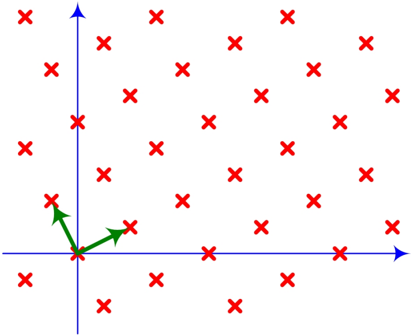
Решётки

- Вот типичная решётка.



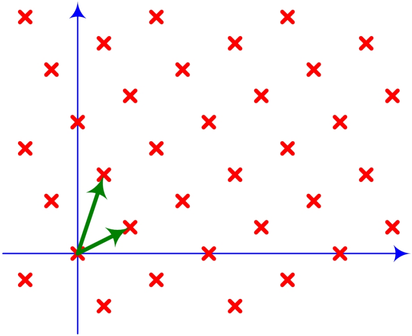
Решётки

- Вот типичная решётка.
- Базис решётки — это множество линейно независимых векторов, её порождающих.



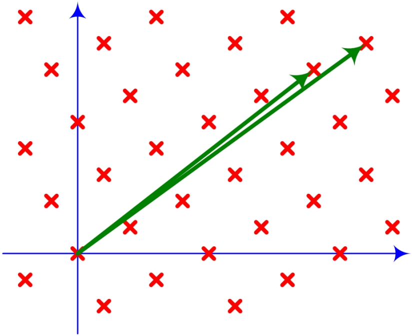
Решётки

- Вот типичная решётка.
- Базис решётки — это множество линейно независимых векторов, её порождающих.
- Вот другой базис той же решётки.



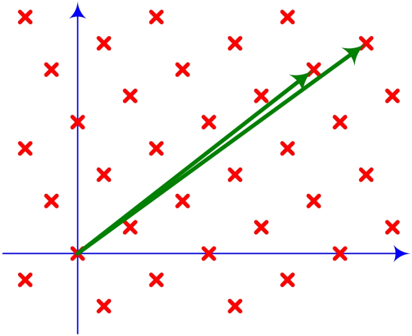
Решётки

- Вот типичная решётка.
- Базис решётки — это множество линейно независимых векторов, её порождающих.
- Вот другой базис той же решётки.
- А вот ещё один.



Решётки

- Базис может состоять из очень длинных векторов, даже если в решётке есть короткие.
- Найти короткий вектор решётки — сложная задача; именно она нам сейчас и понадобится.



Алгоритм L^3

- Мы изучали алгоритм LLL с константным параметром δ, который умеет решать задачу поиска кратчайшего вектора с точностью $\left(\delta - \frac{1}{4}\right)^{-\frac{n-1}{2}}$.
- Можно рассмотреть $\delta = \frac{1}{4} + \left(\frac{3}{4}\right)^{\frac{n}{n-1}}$ и получить точность $\left(\frac{2}{\sqrt{3}}\right)^{n}$.
- Есть ещё алгоритм Шнорра, который достигает субэкспоненциальной точности: $2^{O\left(\frac{n(\log\log n)^2}{\log n}\right)}$.
- Теперь — при чём тут факторизация.

Факторизация через диофантовы приближения

- Пусть нам нужно разложить n.
- Выпишем список простых чисел $p_1, \ldots, p_t$, меньших $(\log n)^\alpha$ (это быстро).
- Предположим, что мы можем найти $t+2$ таких вектора $e = (e_1, \ldots, e_t)^\top \in \mathbb{Z}^t$, что

$$\left| \sum_{i=1}^{t} e_i \log p_i - \log n \right| \le n^{-c} p_t^{o(1)},$$

$$\sum_{i=1}^{t} |e_i \log p_i| \le (2c-1)\log n + 2\log p_t$$

(т.е. они короткие и хорошо приближают $(\log n, \ldots, \log n)$).

Факторизация через диофантовы приближения

- Рассмотрим для каждого вектора $e = (e_1, \ldots, e_t)^\top$ два числа:
$$u = \prod_{e_j > 0} p_j^{e_j}, \qquad v = \prod_{e_j < 0} p_j^{|e_j|}.$$
- Тогда (как показал Шнорр; без д-ва) u хорошо приближает vn:
$$|u - vn| \leq p_t^{1+o(1)},$$
а значит, $|u - vn|$ будет p_t-гладким!
- Разложив, мы найдём (скорее всего) нетривиальное соотношение
$$\prod_{e_j > 0} p_j^{e_j} = \pm \prod_{j=1}^{t} p_j^{b_j} \pmod{n}.$$

Факторизация через диофантовы приближения

- Итак, мы нашли по хорошему вектору соотношение

$$\prod_{e_j>0} p_j^{e_j} = (-1)^{b_0} \prod_{j=1}^{t} p_j^{b_j} \pmod{n}.$$

- Если мы найдём $t+2$ хороших вектора, мы сможем решить систему и найти нетривиальное соотношение по модулю два между векторами экспонент $(a_0, \ldots, a_t) + (b_0, \ldots, b_t)$, где $a_0 = 0$, $a_j = [e_j > 0]e_j$.
- А это позволит нам найти нетривиальное равенство $x^2 \equiv y^2 \pmod{n}$, точно как в методе Крайчика; отсюда и появится разложение n.

Факторизация через решётки

- Но как найти такие приближения?
- Выпишем список простых чисел $p_1, \ldots, p_t$, меньших $(\log n)^\alpha$ (это быстро).
- Рассмотрим решётку $L_{\alpha,c} \subset \mathbb{R}^{t+2}$:

$$\begin{pmatrix} \log 2 & 0 & 0 & \ldots & 0 & n^c \log 2 \\ 0 & \log 3 & 0 & \ldots & 0 & n^c \log 3 \\ 0 & 0 & \log 5 & \ldots & 0 & n^c \log 5 \\ \vdots & \vdots & & & \vdots & \\ 0 & 0 & 0 & \ldots & \log p_t & n^c \log p_t \end{pmatrix}$$

Факторизация через решётки

- Затем мы в этой решётке найдём $t+2$ вектора, достаточно близких к
$$(0, 0, \dots, 0, n^c \log n).$$
- Шнорр показал, что можно так подобрать параметры «близости векторов», что они повлекут как раз требуемое разложение.

Алгоритм Видеманна

- И напоследок повторим алгоритм Видеманна с объяснением алгоритма Берлекампа-Месси (потому что не успели разобрать Берлекампа-Месси раньше).

Алгоритм Видеманна

- Задача: найти такой вектор $\mathbf{w}$, что $A\mathbf{w} = \mathbf{0}$.
- Рассмотрим случайные векторы $\mathbf{x}$ и $\mathbf{z}$, а также $\mathbf{y} = A\mathbf{z}$. Рассмотрим

$$\mathbf{x}^\top \mathbf{y}, \mathbf{x}^\top A\mathbf{y}, \mathbf{x}^\top A^2\mathbf{y}, \mathbf{x}^\top A^3\mathbf{y}, \ldots$$

Минимальный многочлен

- Вспомним линейную алгебру: у матрицы A размера $n \times n$ есть минимальный многочлен p степени $n_0 \leq n$, для которого $p(A) = 0$.
- Пусть минимальный многочлен p: $\sum_{i=0}^{n_0} p_i A^i = 0$. Значит,

$$\sum_{i=0}^{n_0} p_i \mathbf{x}^\top A^i \mathbf{y} = 0,$$

  и этот многочлен также порождает и нашу последовательность.
- Как найти порождающий многочлен?

Алгоритм Берлекампа-Месси

- Рассмотрим последовательность s длины $n+1$: $s^{n+1} = s_0 s_1 \ldots s_{n-1} s_n$.
- Пусть $\langle L, C(D) \rangle$, $C(D) = 1 + c_1 D + \ldots + c_L D^L$, порождает $s^n = s_0 s_1 \ldots s_{n-1}$.
- Рассмотрим разницу

$$d_n = s_n \oplus \sum_{i=1}^{L} c_i s_{N-i}.$$

Алгоритм Берлекампа-Месси

- Если $d_n = 0$, всё хорошо, берём $L(s^{n+1}) = L$.
- Если $d_n = 1$, рассмотрим предыдущий LFSR, который отличался, т.е. максимальное такое $m < n$, что $L(s^m) < L(s^n)$. Пусть это был $\langle L(s^m), B(D) \rangle$.
- Теперь, если $L > n/2$, то $L' = L$, а если $L \leq n/2$, то $L' = n + 1 - L$.
- И результат — $\langle L', C'(D) \rangle$, $C'(D) = C(D) + B(D) \cdot D^{n-m}$.

Упражнение. Доказать, что действительно получается многочлен, порождающий s^n.

Алгоритм Видеманна

- Итак, мы применяем алгоритм Берлекампа-Месси и получаем такие коэффициенты q_i, что

$$\sum_{i=0}^{n_0} q_i \mathbf{x}^\top A^i \mathbf{y} = 0.$$

- Мы надеемся, что при этом заодно и

$$\sum_{i=0}^{n_0} q_i A^i \mathbf{y} = 0, \text{ и, т.к. } \mathbf{y} = A\mathbf{z}, \quad M\left(\sum_{i=0}^{n_0} q_i A^i \mathbf{z}\right) = 0,$$

и мы надеемся, что $\mathbf{w} = \sum_{i=0}^{n_0} q_i A^i \mathbf{z} \neq 0$, ведь тогда это и есть решение.

- Наши надежды часто (по $\mathbf{x}$ и $\mathbf{y}$) будут оправдываться.

Спасибо за внимание!

- Lecture notes и слайды будут появляться на моей homepage:
  http://logic.pdmi.ras.ru/~sergey/
- Присылайте любые замечания, решения упражнений, новые численные примеры и прочее по адресам: sergey@logic.pdmi.ras.ru, snikolenko@gmail.com.