Лекция 5 от 27.09.2016. RSA, продолжение некоторых теоретико числовых алгоритмов, некоторые комбинаторные оптимизации

> «На самом деле, если этот алгоритм (разложение на простые) придумают за полиномиальное время, можно спокойно идти и покупать попкорн и смотреть, как рушится этот мир.
> Только не забудьте перед этим снять все деньги с банковских карточек.»
>
> Глеб

Прим. Я в этой лекции поменял местами темы, чтобы было легче воспринимать материал.

Предисловие

Теория чисел с появлением алгоритмов, а особенно криптографии, приобрела новую «жизнь». Теперь простые числа, разложение на простые множители являются важными алгоритмами даже в нашей повседневной жизни. Сначала поговорим о самих простых числах, потом о криптографии.

Тест Рабина-Миллера на проверку простоты числа

В той же самое криптографии есть необходимость в генерировании длинных простых чисел. Благо простые числа встречаются не так редко. Пусть функция распределения простых чисел будет $\pi(n)$ — количество чисел, не бóльших n, которые являются простыми. Есть теорема, о которой мы уже упоминали

$$\lim_{n\to+\infty} \frac{\pi(n)}{n/\ln(n)} = 1$$

То есть если взять случайно $\ln(n)$ чисел до n, то математическое ожидание, что хотя бы одно из них будет простым равно 1.

Поэтому, чтобы сгенерировать большое простое число, надо уметь проверять за полином от числа (было бы вообще прекрасно), является ли число простым. Был найден алгоритм, который проверяет это свойство за полином от размера числа.

Но мы рассмотрим достаточно эффективный вероятностный алгоритм проверки числа на простоту. Вспомним малую теорему Ферма, которая нам гласит:

$$a^{p-1} \equiv 1 \bmod p \text{ для всех простых чисел } p.$$

Определение 1. *Назовём число n псевдопростым по основанию a, если $(a, n) = 1$ и $a^{n-1} \equiv 1 \pmod{n}$.*

Поэтому первый наш алгоритм будет такой (будем проверять нечётные n и $a = 2$):

Algorithm 1 Проверка на псевдопростоту по основанию 2.

```
1: function PSEUDOPRIME(int n)
2:     if FAST_POW(2, n − 1, n) ≠ 1 then
3:         return Составное                          ▷ 100% составное
4:     return Простое          ▷ молимся и надеемся, что тут действительно простое
```

Но, к сожалению, существуют и составные числа, которые удовлетворяют этому алгоритму и нам выведется, что число простое, а на самом деле нет. Наименьшие из них это $341, 561 \ldots$ и так далее. Поменять основание тоже не вариант, так как существуют числа Карлмайка, которые псевдопростые по любому основанию. Но было доказано, что с ростом n вероятность, что непростое число является псевдопростым стремится к 0.

К ~~счастью~~ сожалению, мы не будем доказывать вероятность ошибки этого алгоритма (она составляет не более 0.5), поэтому если запустить этот алгоритм k раз, то вероятность ошибки будет равна 2^{-k}. Собственно, алгоритм:

Algorithm 2 Тест Миллера-Рабина, считаем, что $n - 1 = 2^t u$, где u нечетно и $t \geqslant 1$

```
1: function MILLER_RABIN(int n)
2:     a ← random_integer(1, n − 1)
3:     x_0 ← FAST_POW(a, u, n)
4:     for i ← 1 to t do
5:         x_i ← x_{i−1}^2 mod n
6:         if x_i = 1 и x_{i−1} ≠ 1 и x_{i−1} ≠ n − 1 then
7:             return Составное
8:     if x_t ≠ 1 then
9:         return Составное
10:    return Простое
```

Ясно, что этот алгоритм работает за полином от числа (при условии, что мы умеем вычислять по модулю быстро, но это задача в листке, да и это легко придумать без метода Ньютона, оставляю читалю это сделать).

Лемма 1. *Если алгоритм как-то вышел на строчки 7 или 9, то число действительно составное.*

Доказательство. Заметим, что $x_i \equiv a^{2^i u} \bmod n$, так как $x_0 \equiv a^u \bmod n$ (база индукции) и $x_i = x_{i-1}^2 \bmod n$ (переход индукции), а значит $x_i \equiv a^{2^{i-1}u} \cdot a^{2^{i-1}u} \equiv a^{2^i u} \bmod n$.

Поэтому если $x_i = 1$ и $x_{i-1} \neq 1$ и $x_{i-1} \neq n - 1$, то $n \mid x_{i-1}^2 - 1$, то есть $(x_{i-1} - 1)(x_{i-1} + 1) = 0$ в $\mathbb{Z}_n$. То есть у нас нетривиальные делители нуля, а из курса алгебры известно, что в поле (а при простых n — $\mathbb{Z}_n$ поле) их нет, поэтому перед нами составное число.

Если $x_t \neq 1$, то просто-напросто не выполняется малая теорема Ферма и тогда n точно составное. □

Лемма 2. *Количество таких a, на которых алгоритм выдаст «Составное» не меньше $\frac{n-1}{2}$ при составном нечетном n.*

Именно этот факт мы оставим без доказательства (*Прим. на самом деле, он не очень сложный, видимо, Глебу было лень*). И именно он нам даёт ошибку не более 0.5.

RSA, криптография

Криптографическую систему с открытым ключом можно использовать для шифровки сообщений, которыми обмениваются 2 партнера (Алиса и Боб), чтобы посторонние люди (Ева в дальнейшем), даже перехватившие сообщения, не могли его расшифровать. Также некоторая система позволяет подписывать свои подписи. Кто угодно без труда может её проверить, но подделать никак.

Давайте уже перейдём к обсуждению различных систем.

Но для начала несколько определений. У Алисы есть ключи P_A, R_A, у Боба P_B, R_B — публичные и приватные соответственно (на самом деле это функции, которые что-то вычисляют). Алиса и Боб хранят приватные ключи у себя, а с открытыми можно делать что угодно. Будем считать, что Алиса и Боб передают двоичные последовательности. Также будем считать, что $M = P_A(R_A(M)) = R_A(P_A(M))$ и $M = P_B(R_B(M)) = R_B(P_B(M))$. Также, чтобы шифрование имело смысл, надо, чтобы секретные ключи владельцы умели вычислять быстро, и чтобы по открытому ключу было очень сложно вычислить обратное преобразование. На этом и держится весь алгоритм. Рассмотрим пример:

Боб хочет отправить сообщение M Алисе, зашифрованное так, чтобы для Евы оно выглядело как ужасный набор символов:

- Боб получает открытый ключ Алисы P_A любым способом;
- Боб шифрует сообщение, которое знает только он, как $C = P_A(M)$;
- Алиса, когда получает сообщение C, расшифровывает своё сообщение с помощью секретного ключа.

Функции обратные, поэтому вычисления будут корректными. Но, к сожалению, такая система плоха тем, что, перехватив сообщения, Ева может их подменивать. Поэтому часто используют ещё и цифровые подписи:

Пусть Алиса хочет отправить сообщение M' Бобу:

- Алиса вычисляет свою подпись с помощью своего секретного ключа. $\sigma = R_A(M')$;
- Алиса отправляет пару Бобу (M', σ);
- Боб может легко убедиться, что это действительно Алиса, с помощью открытого ключа, вычислив $P_A(\sigma)$ и сравнив с M'.

В данном случае никакая Ева не страшна в подмене сообщения, так как она не может вычислить $R_A(M')$ ни для какого M'.

Такие подписи позволяют проверять целостность сообщений. Но всё равно есть проблема — Ева знает содержания сообщений. Можно взять ещё 1 ключ, который шифрует по 1-ой схеме сообщения, которые мы передаём по 2-ой схеме. И тогда Ева, даже получив перехваченное сообщение, во-первых, не сможет понять, какой парой оно было зашифровано, то есть дешифровка невозможна за разумное время, да и подмена тоже, так как там применяется к сообщению сложный ключ.

Проблема остаётся одна — что Алиса и Боб должны обменяться ключами, чтобы Ева не могла подменить открытые ключи. Но, к сожалению, невозможно спрятаться от Евы, если быть совсем параноиком. Она всегда может подменять вам ключи, где бы вы ни находились. Поэтому фактор личной встречи должен быть. И самая большая «insecurity» состоит именно в том, что Ева внедряется работать к Алисе, чтобы разузнать, а то и подменять ключи для Боба.

Давайте уже, наконец-то поговорим о способах шифрования:

- Самая старая система это шифр Цезаря. Она просто переставляет по циклу символы в алфавите, что конечно, же ломается за $\mathcal{O}(k)$, где k — размер алфавита
- Взять рандомную перестановку алфавита. Да, задача уже сложнее, но если это какой-нибудь язык, то можно из статистических параметров восстанавливать символы, что значительно сократит перебор. Не годится.
- Например, выравнивать двоичные сообщения и брать рандомную перестановку, применяя её ко всем сообщениям. Тогда нетрудно убедиться, что за $\mathcal{O}(\log n)$ действий мы сможем понять, какой символ, где стоит. Просто смотреть, куда переходят единицы и нули. На непонятных рандомных сообщениях математическое ожидание того, чтобы разобрать, где что стоит, будет $\mathcal{O}(\log n)$.

Все примеры сверху так или иначе зависят от человеческого фактора или для них легко подобрать обратную функцию. Рассмотрим криптографическую систему RSA (Rivest–Shamir–Adleman public-key cryptosystem).

1. Случайным образом выбираются 2 больших простых числа $p \neq q$. Мы уже обсудили выше, что это сделать легко достаточно.
2. Вычисляется $n = pq$ (что можно сделать тоже не очень сложно алгоритмом Карацубы или преобразованием Фурье).
3. Выбирается маленькое нечетное число e, взаимно простое с $\varphi(n) = (p-1)(q-1)$ из-за мультипликативности.
4. Вычисляем число $d = e^{-1} \bmod \varphi(n)$. Это можно сделать расширенным алгоритмом Евклида.
5. Пара $P = (e, n)$ будет открытым ключом.
6. Пара $S = (d, n)$ закрытым.

Теперь в качестве сообщения мы передаём сообщение $P(M) = M^e \bmod n$.

Обратное шифрование будет равно $S(C) = C^d \bmod n$.

Аналогично, ясно, что это шифрование работает за полином от длины числа, так как все операции мы умеем делать за полином от числа.

Лемма 3 (Корректность RSA)**.** *Докажем, что это взаимно обратные функции.*

Доказательство. Видно, что $P(S(M)) = S(P(M)) = M^{ed} \bmod n$.

Так как e, d взаимно обратные по модулю $\varphi(n)$, то $ed = 1 + k(p-1)(q-1)$

$$M^{ed} \equiv M^{1+k(p-1)(q-1)} \equiv M \cdot M^{k(q-1)\varphi(p)} \bmod p$$

$$M^{ed} \equiv M^{1+k(p-1)(q-1)} \equiv M \cdot M^{k(p-1)\varphi(q)} \bmod q$$

Малая теорема Ферма имеет очень простое следствие, что для любых чисел $M^p \equiv M \bmod p$ (предлагается это доказать самостоятельно). Поэтому в обоих равенствах в арифметике это просто эквивалентно M, то есть

$$M^{ed} \equiv M \bmod p \text{ и } M^{ed} \equiv M \bmod q$$

А значит по легкому следствию из основной теоремы арифметики $M^{ed} \equiv M \bmod n$. □

Основная сложность в том, что зная $n, e, M^e \bmod n$ практически невозможно найти M. Перебрать все M может занять экспоненциальную сложность, а разложение на множители n и вычисление d оказалось очень сложной задачей, которая пока не решается за полиномиальное время.

Байка от Глебаса: На самом деле вся теория по шифрованию в интернете появилась лет 5-6 назад. Раньше кто угодно мог перехватывать сообщения вашей почты, платить в интернете было опасно (если вообще можно было). Я только однажды покупал что-то не через безопасное соединение в интернете. Я очень хотел ту пиццу, мне было без разницы тогда на безопасность.

Комбинаторные оптимизации. Генетический алгоритм

На этой лекции был треш. Я честно не знаю, как это конспектировать. Напишу то, в чём я разобрался.

На лекции были рассмотрены метод Ньютона, отжига и генетический алгоритм. Всегда есть Google, поэтому первые 2 ищите там.

Генетический алгоритм позволяет решать некоторые трудные задачи методом ошибок за разумное время.

Есть несколько фаз алгоритма:

- **Создание популяции.** Обычно это какие-то рандомные решение нашей задачи, которые могут иметь много ошибок.
- **Размножение.** Тут всё как у людей. Мы скрещиваем некоторые особи (обычно сильные особи) вместе, чтобы получить лучшее поколение.
- **Мутация.** Тут природа говорит, что мутации иногда хорошо влияют на организмы. Мы просто берем некоторые особи и мутируем их с помощью какого-то заранее определенного алгоритма. Да, могут получиться плохие особи, но есть вероятность, что получатся хорошие.
- **Отбор.** Мы отбираем самых лучших, те, кто пойдут дальше повторять этот процесс.

Иногда такой алгоритм приносит правильные решения.

Далее материала не было на лекции.

Приведём пример работы генетического алгоритма в задаче о правильной расстановке ферзей.

Берем какую-нибудь перестановку, что в строках и столбцах ровно по 1 ферзю. Генерируем, например, 100 таких перестановок.

Потом считаем количество ошибок, то есть количество пар, которые бьют друг друга (это можно сделать за $\mathcal{O}(n)$, пройдясь по диагоналям).

После этого выбираем хороших особей — примерно половину, у которых ошибок меньше всего. Берем любые 2, смотрим, какие элементы у них совпадают, оставляем их в предположении, что они являются «сильными» генами, а остальное всё перемешиваем между собой. Так делаем с каким-то количеством пар, потом выбираем, например, 1 особь, мутируем её — меняем 2 любых элемента местами.

Опять же считаем количество ошибок и выбираем лучших 100. Повторяем алгоритм, пока не найдём нужное решение.

Генетический алгоритм ничего не доказывает, он лишь может работать как природа. Мы можем находить какие-то хорошие решение с его помощью за более разумное время.

Байка от Глебаса: В конце 40-ых годов появилась компания RAND, которая одна из первых вообще придумала работать со случайными числами (*Прим. и даже сгенерировала огромный список случайных цифр*). Вообще, это была первая компания, которая моделировала процессы с помощью случайных чисел.