Математические основы защиты информации
Лабораторная работа №6
Факторные базы.

БГУ, ММФ, Каф. ДУиСА,
доцент Чергинец Д.Н.

Факторные базы

Большинство современных методов факторизации чисел основаны на поиске чисел $s, t \in \mathbb{Z}_n$, что $s^2 \equiv t^2 \pmod{n}$.
Если такие числа найдены и $s \not\equiv \pm t \pmod{n}$, то $1 < \mathrm{GCD}(s \pm t, n) < n$. То есть мы нашли нетривиальный делитель $d = \mathrm{GCD}(s \pm t, n)$.

Факторной базой называется конечное множество различных простых чисел $\mathrm{B} = \{p_1, p_2, \ldots, p_m\}$ (за исключением p_1, которое иногда выбирают равным -1). Как правило, берут подряд идущие простые числа, не превосходящие некоторой границы M, например,

$$B = \{2, 3, 5, 7, 11, 13\} \qquad \text{или} \qquad B = \{-1, 2, 3, 5, 7, 11, 13\}.$$

Целое число a называется ***B*-гладким (или *M*-гладким)**, если оно раскладывается в произведение чисел из факторной базы B:

$$a = p_1^{\alpha_1} \ldots p_m^{\alpha_m}, \quad p_i \in B.$$

Каждому B-гладкому числу a сопоставим вектор, состоящий из степеней $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$.

Факторизация чисел

Алгоритм Диксона факторизации чисел

Рассмотрим на примере работу алгоритма Диксона.

```
In[ ]:= n = 4 291 753
Out[ ]= 4 291 753
```

```
In[ ]:= B = {2, 3, 5, 7}
        m = Length[B]
              длина
Out[ ]= {2, 3, 5, 7}
Out[ ]= 4
```

Символом $R(b)$ будем обозначать наименьший неотрицательный вычет в классе $b^2 \pmod n$. То есть $R(b) := \mathrm{Mod}[b^2, n]$.
Методом пробных делений находим такие случайные числа $b_1, b_2, \ldots, b_k$, $k := m + 1$, что $R(b_1), \ldots, R(b_k)$ являются B-гладкими. Также вычисляем списки α_i степеней

```
In[ ]:= k = m + 1;
        b = {992 736, 1 627 156, 3 649 873, 1 856 523, 64}
        α = {{8, 1, 2, 2}, {6, 5, 2, 1}, {6, 1, 2, 3}, {2, 0, 4, 3}, {12, 0, 0, 0}}
        FactorInteger[Mod[b[[1]]^2, n]]
        факторизовать ⋯ остаток от деления
Out[ ]= {992 736, 1 627 156, 3 649 873, 1 856 523, 64}
Out[ ]= {{8, 1, 2, 2}, {6, 5, 2, 1}, {6, 1, 2, 3}, {2, 0, 4, 3}, {12, 0, 0, 0}}
Out[ ]= {{2, 8}, {3, 1}, {5, 2}, {7, 2}}
```

Далее методом Гаусса решаем однородную алгебраическую систему $x_1\,\alpha_1 + \ldots + x_k\,\alpha_k = 0$ над полем $x_i \in \mathbb{Z}_2$. Данная система всегда имеет решения. В нашем случае решение имеет вид

```
In[ ]:= xx = Array[x, k]
             массив
        sol = Solve[xx.α == 0, xx, Modulus → 2][[1]]
              решить уравнения        модуль
Out[ ]= {x[1], x[2], x[3], x[4], x[5]}
Out[ ]= {x[1] → c2, x[2] → c2 + c3, x[3] → c3, x[4] → c2, x[5] → c1}
```

В качестве рекламы дисциплин по выбору отметим, что обобщением метода Гаусса на случай нелинейных систем являются так называемые Базисы Грёбнера, при помощи которых
однородная система

In[•]:=
```
xx.α
```

Out[•]=
```
{8 x[1] + 6 x[2] + 6 x[3] + 2 x[4] + 12 x[5], x[1] + 5 x[2] + x[3],
 2 x[1] + 2 x[2] + 2 x[3] + 4 x[4], 2 x[1] + x[2] + 3 x[3] + 3 x[4]}
```

приводится к гораздо более простой

In[•]:=
```
GroebnerBasis[xx.α, xx, Modulus → 2]
```
базис Грёбнера | модуль

Out[•]=
```
{x[2] + x[3] + x[4], x[1] + x[4]}
```

Пространство решений содержит три произвольных постоянных

In[•]:=
```
xxx = xx /. sol
VarsParam = Variables[xxx]
dim = Length[VarsParam]
```
переменные | длина

Out[•]= $\{c_2, c_2 + c_3, c_3, c_2, c_1\}$

Out[•]= $\{c_1, c_2, c_3\}$

Out[•]= 3

Получили 2^3 решений системы. Возьмем первое ненулевое решение

In[•]:=
```
subs = Thread[VarsParam → IntegerDigits[1, 2, dim]]
Sol1 = Mod[xxx /. subs, 2]
```
нанизать | цифры целого числа | остаток от деления

Out[•]= $\{c_1 \to 0, c_2 \to 0, c_3 \to 1\}$

Out[•]= {0, 1, 1, 0, 0}

Мы получили,

что

$$(b_1^{x_1} b_2^{x_2} \ldots b_k^{x_k})^2 \equiv \left(p_1^{\sum_{i=1}^k x_i \alpha_{i,1}} \ldots p_m^{\sum_{i=1}^k x_i \alpha_{i,m}}\right) \equiv \left(p_1^{\frac{1}{2}\sum_{i=1}^k x_i \alpha_{i,1}} \ldots p_m^{\frac{1}{2}\sum_{i=1}^k x_i \alpha_{i,m}}\right)^2 \pmod{n}.$$

Поэтому

$$s = b_1^{x_1} b_2^{x_2} \ldots b_k^{x_k}$$

$$t = p_1^{\frac{1}{2}\sum_{i=1}^k x_i \alpha_{i,1}} \ldots p_m^{\frac{1}{2}\sum_{i=1}^k x_i \alpha_{i,m}}$$

И если $s \not\equiv \pm t \pmod{n}$, то находим два нетривиальных делителя $d_{1,2} = \mathrm{GCD}(s \pm t, n)$.

In[•]:=
```
s = Mod[Times @@ b^Sol1, n]
```
ос… | умножить

Out[•]= 2116800

In[•]:=
```
stepen = Plus @@ (α Sol1) / 2
```

Out[•]= {6, 3, 2, 2}

```
In[•]:= t = Mod[Times @@ B^stepen, n]
             ос⋯ умножить
```

```
Out[•]= 2 116 800
```

Полученные $t = 2116800$ и $s = 2116800$ не подходят ($t = s$), но если n не является степенью простого числа $n \neq p^{\alpha}$, то "хороших" чисел t и s не меньше половины. Попробуем взять другое решение

```
In[•]:= subs = Thread[VarsParam → IntegerDigits[2, 2, dim]]
               нанизать            цифры целого числа
        Sol2 = Mod[xxx /. subs, 2]
               остаток от деления
Out[•]= {ℂ1 → 0, ℂ2 → 1, ℂ3 → 0}
Out[•]= {1, 1, 0, 1, 0}
```

```
In[•]:= s = Mod[Times @@ b^Sol2, n]
             ос⋯ умножить
        stepen = (Plus @@ (α Sol2))/2
        t = Mod[Times @@ B^stepen, n]
             ос⋯ умножить
Out[•]= 3 223 326
Out[•]= {8, 3, 4, 3}
Out[•]= 1 105 215
```

```
In[•]:= d1 = GCD[t + s, n]
             НОД
        d2 = GCD[t - s, n]
             НОД
Out[•]= 541
Out[•]= 7933
```

Таким образом, мы нашли два нетривиальных делителя числа n.

Задание 1.

Реализовать алгоритм Диксона, использующий в качестве представителя класса $b^2 \pmod n$ наименьший неотрицательный вычет $R(b)$, включающий в себя следующие процедуры:

1) процедура, вычисляющая факторную базу $B := \{p_1, p_2, \ldots, p_m \mid p_i - \text{простое}, \ p_i < M\}$, где $M := {L_n}^c$, $L_n := \mathrm{Exp}\left[(\mathrm{Log}[n]\,\mathrm{Log}[\mathrm{Log}[n]])^{1/2}\right]$, c – входной параметр алгоритма, $\frac{1}{2} \leq c \leq 1$.

2) процедура, тестирующая число a на B-гладкость для данной факторной базы B.

3) процедура, методом пробных делений вычисляющая случайные числа $b_1, b_2, \ldots, b_k, \quad k = m + 1,$ для которых $R(b_i)$ является B-гладким числом. Вычисляются соответствующие им степени $\alpha_1, \alpha_2, \ldots, \alpha_k$.

4) процедура, вычисляющая делители числа n, используя

- факторную базу B,
- числа $b_1, b_2, \ldots, b_k$ и соответствующие им степени $\alpha_1, \alpha_2, \ldots, \alpha_k$.

Задание 2.

При помощи функции Timing определите значение параметра c, при котором

a) для числа n_1;

b) для числа n_2

тратится меньше всего времени на

1) вычисление чисел $b_1, b_2, \ldots, b_k$ ($R(b_i)$ является B-гладким);

2) работу алгоритма Диксона.

Вариант 1.

```
n1 = 295 927;
n2 = 829 459 817;
```

Вариант 2.

```
n1 = 664 889;
n2 = 1 779 739 817;
```

Вариант 3.

```
n1 = 1 078 213;
n2 = 2 774 667 139;
```

Вариант 4.

```
n1 = 1 487 209;
n2 = 3 800 636 941;
```

Вариант 5.

```
n1 = 1 937 321;
n2 = 4 846 087 483;
```

Вариант 6.

```
n1 = 2 391 761;
n2 = 5 913 727 063;
```

Вариант 7.

```
n₁ = 2 857 021;
n₂ = 6 987 638 491;
```

Вариант 8.

```
n₁ = 3 323 363;
n₂ = 8 080 460 491;
```

Вариант 9.

```
n₁ = 3 787 541;
n₂ = 9 182 325 989;
```

Вариант 10.

```
n₁ = 4 288 507;
n₂ = 10 292 490 599;
```

Вариант 11.

```
n₁ = 4 780 817;
n₂ = 11 411 682 869;
```

Вариант 12.

```
n₁ = 5 268 799;
n₂ = 12 540 457 129;
```

Вариант 13.

```
n₁ = 5 768 683;
n₂ = 13 675 724 969;
```

Вариант 14.

```
n₁ = 6 317 257;
n₂ = 14 813 827 811;
```

Вариант 15.

```
n₁ = 6 799 829;
n₂ = 15 958 202 501;
```

Символом $A(b)$ будем обозначать наименьший по абсолютному значению вычет в классе $b^2 \pmod n$.

$$A(b) := \begin{cases} R(b), & \text{если } R(b) < \frac{n}{2}; \\ R(b) - n, & \text{если } R(b) > \frac{n}{2}. \end{cases}$$

Задание 3.

Реализовать алгоритм Диксона, использующий числа $b_1, \ldots, b_k$, для которых $A(b_i)$ является B-гладким числом. При формировании факторной базы первым элементом нужно взять -1 ($p_1 = -1$). Сравнить его время вычисления с алгоритмом Диксона, использующим наименьшие неотрицательные вычеты.

Быстрая проверка на B-гладкость

Есть более быстрый способ проверки числа на B-гладкость.
Пусть
$B = \{p_1, p_2, \ldots, p_m\}$ — факторная база.
n — факторизуемое число.
Если $p_1 \neq -1$, то checker := $p_1^{\text{Floor[Log}[p_1,n]]} \ldots p_m^{\text{Floor[Log}[p_m,n]]}$.
Если $p_1 == -1$, то checker := $p_2^{\text{Floor[Log}[p_2,n/2]]} \ldots p_m^{\text{Floor[Log}[p_m,n/2]]}$.

Тогда
b — B-гладкое $\Leftrightarrow$ НОД$(b, \text{checker}) = |b|$.

Задание 4.

Реализовать алгоритм Диксона, используя быструю проверку на B-гладкость. Стал ли алгоритм работать быстрее?

Оценка сложности вычислений

Пусть n — входные данные алгоритма,
$f(n)$ — количество арифметических операций, необходимых для выполнения всех действий алгоритма с входными данными n. Временной сложностью вычисления алгоритма называется функция
$T(N) := \max\limits_{<n> \leq N} f(n)$.
Где $<n>$ - количество бит в двоичной записи входных данных n.

Алгоритм называется полиномиальным, если его сложность $T(N)$ и длина чисел, участвующих в промежуточных вычислениях, ограничены полиномом от N.

Например, алгоритм со временем вычислений $T(N) = C\,N$ является полиномиальным, и так как сложность является линейной функцией, то и алгоритм называют линейным.

Если $T(N) = 2^{O(N)}$, то алгоритм называют экспоненциальным.
Если $T(N) = 2^{o(N)}$, то алгоритм называют субэкспоненциальным.

Пусть
$L_n[\alpha, c] := \mathrm{Exp}\left[(c + o(1))\,(\ln n)^{\alpha}\,(\ln\ln n)^{1-\alpha}\right]$
Для алгоритма Диксона справедливо равенство
$f(n) = L_n\left[1/2,\, 2\sqrt{2}\right]$.

Задание 5.

Используя равенство
$f(n) = L_n\left[1/2,\, 2\sqrt{2}\right]$,
доказать, что алгоритм Диксона является субэкспоненциальным.