Алгоритмы и структуры данных

Получисленные алгоритмы

д.т.н., проф. Трифонов Петр Владимирович

Содержание лекции

1. Быстрое умножение многочленов
2. Быстрое преобразование Фурье
3. Операции над целыми числами
4. Расширенный алгоритм Евклида
5. Модулярное умножение
6. Возведение в степень
7. Криптография с открытым ключом
8. Операции над матрицами

Получисленные алгоритмы

- Вычислительные примитивы (умножение, элементарные преобразования)
- Как правило, основаны на стратегии "разделяй и властвуй"
- Д. Кнут. Искусство программирования, Т.2: Получисленные алгоритмы

Линейная и циклическая свертки

Определение

Пусть даны векторы $(a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{n-1})$ или соответствующие многочлены $a(x) = \sum_{i=0}^{n-1} a_i x^i, b(x) = \sum_{i=0}^{n-1} b_i x^i$. Их линейной сверткой называется вектор $(c_0, \ldots, c_{2n-2})$, соответствующий многочлену
$c(x) = a(x)b(x) = \sum_{i=0}^{2n-2} c_i x^i, c_i = \sum_{j=\max(0,i-(n-1))}^{\min(i,n-1)} a_{i-j} b_j$

Определение

Пусть даны векторы $(a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{n-1})$ или соответствующие многочлены $a(x) = \sum_{i=0}^{n-1} a_i x^i, b(x) = \sum_{i=0}^{n-1} b_i x^i$. Их циклической сверткой называется вектор $(c_0, \ldots, c_{n-1})$, соответствующий многочлену $c(x) \equiv a(x)b(x) \bmod (x^n - 1)$, где
$c(x) = \sum_{i=0}^{n-1} c_i x^i,\ c_i = \sum_{j=0}^{n-1} a_{(i-j) \bmod n} b_j$

Непосредственное вычисление требует n^2 умножений и $\Theta(n^2)$ сложений

Интерполяционный полином Лагранжа

- Пусть даны точки $(x_i, y_i), 0 \leq i < n$, где x_i — различные значения
- Найти многочлен $f(x)$ наименьшей степени, т.ч. $f(x_i) = y_i$

$$f(x) = \sum_{i=0}^{n-1} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

Алгоритм Тоома-Кука

- Рассмотрим быстрое вычисление $c(x) = a(x)b(x), \deg a(x), b(x) < n$
- Выберем различные x_i. Вычислим $A_i = a(x_i), B_i = b(x_i), 0 \le i < 2n-1$
- Вычислим $c(x_i) = C_i = A_i B_i$
- $c(x) = \sum_{i=0}^{n-1} C_i \underbrace{\prod_{j\neq i} \frac{x - x_j}{x_i - x_j}}_{\text{вычислить заранее}}$
- x_i надо выбрать так, чтобы коэффициенты в $c(x)$ получались "вычислительно простыми"
- Пример: $c_0 + c_1 x + c_2 x^2 = (a_0 + a_1 x)(b_0 + b_1 x), x_0 = -1, x_1 = 0, x_2 = 1$

  $C_0 = a(-1)b(-1) = (a_0 - a_1)(b_0 - b_1), C_1 = a(0)b(0) = a_0 b_0, C_2 = a(1)b(1) = (a_0 + a_1)(b_0 + b_1)$

  $$c(x) = C_1(1 - x^2) + \frac{1}{2}C_2(x^2 + x) + \frac{1}{2}C_0(x^2 - x) = C_1 + x\frac{1}{2}(C_2 - C_0) + x^2\frac{1}{2}(C_2 + C_0 - 2C_1)$$
- Алгоритм ТК: 3 умножения, 7 сложений. По определению: 4 умножения, 1 сложение

Алгоритм Карацубы

- Рассмотрим вычисление $c(x) = (a_0 + a_1 x)(b_0 + b_1 x)$
- $c'(x) = c(x) - c_2 x^2 = (a_0 + a_1 x)(b_0 + b_1 x) - a_1 b_1 x^2$
- $C'_0 = c'(0) = a_0 b_0$, $C'_1 = c'(1) = (a_0 + a_1)(b_0 + b_1) - a_1 b_1$
- $c'(x) = -C'_0(x-1) + C'_1 x$

$$c(x) = (a_0 + a_1 x)(b_0 + b_1 x) = a_0 b_0 + x((a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0) + x^2 a_1 b_1$$

- Алгоритм Карацубы: 3 умножения, 4 сложения. По определению: 4 умножения, 1 сложение
- Циклическая свертка: $(a_0 + a_1 x)(b_0 + b_1 x) \equiv (a_0 b_0 - a_1 b_1) + x((a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0) \bmod (x^2 - 1)$

Итерированный алгоритм Карацубы: разделяй и властвуй

- Рассмотрим вычисление $c(x) = a(x)b(x), \deg a(x) < n, \deg b(x) < n$
- $a(x) = a_0(x) + x^{n/2}a_1(x), b(x) = b_0(x) + x^{n/2}b_1(x)$

$$a(x)b(x) = a_0(x)b_0(x)+x^{n/2}((a_0(x)+a_1(x))(b_0(x)+b_1(x))-a_0(x)b_0(x)-a_1(x)b_1(x))+x^n a_1(x)b_1(x)$$

- Сложность $T(n) = 3T(n/2) + (3n-2) = \Theta(n^{\log_2 3})$

Нижние границы на сложность умножения многочленов

1. Никакой алгоритм перемножения многочленов степени $L-1$ и $N-1$ не может содержать число умножений меньшее, чем $L+N-1$.
2. Если число простых делителей многочлена $p(x)$ равно t, никакой алгоритм перемножения многочленов (достаточно большой степени n) по модулю $p(x)$ не может содержать число умножений меньшее, чем $2n-t$.

Ряд и преобразование Фурье

- Пусть дана функция $f(x)$, периодическая на интервале $(-\pi, \pi)$.
- Ряд Фурье

$$f(x) = \sum_{k=-\infty}^{\infty} F_k e^{ikx},$$

где

$$F_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{-ikt} dt.$$

- Допускается обобщение на случай функций с произвольным периодом L
- При $L \to \infty$ ряд Фурье переходит в преобразование Фурье $F(t) = \int_{-\infty}^{\infty} f(x) e^{-2\pi itx} dx$
- Обратное преобразование Фурье: $f(x) = \int_{-\infty}^{\infty} F(t) e^{2\pi itx} dt$

Дискретное преобразование Фурье

- Пусть $f(t)$ задана своими отсчетами $f_k = f(k\Delta t), k = 0..n-1$.
- Дискретное преобразование Фурье последовательности f_k: $F_j = \sum_{k=0}^{n-1} f_k e^{-2\pi ijk/n}$.
- Обратное дискретное преобразование Фурье $f_k = \frac{1}{n}\sum_{j=0}^{n-1} F_j e^{2\pi ikj/n}$
- Встречаются различные варианты распределения множителя $1/n$, и -1 в показателе степени
- Ядро ДПФ $\alpha = e^{-2\pi i/n}$. В более общем случае, α — некоторый элемент порядка n

$$
\begin{aligned}
F_j &= \sum_{k=0}^{n-1} f_k \alpha^{jk} \\
f_k &= \frac{1}{n}\sum_{j=0}^{n-1} F_j \alpha^{-jk}
\end{aligned}
$$

Корректность обратного преобразования Фурье

- $\alpha^n = 1 \Rightarrow \alpha^{rn} - 1 = 0$
- Над любым полем $x^n - 1 = (x-1)(x^{n-1} + \ldots + 1) = 0$
- Следовательно, $0 = \alpha^{rn} - 1 = (\alpha^r - 1)(\alpha^{r(n-1)} + \alpha^{r(n-2)} + \cdots + 1)$
- Для всех $r \not\equiv 0 \bmod n : \alpha^r \neq 1$. Следовательно, $\sum_{j=0}^{n-1} \alpha^{rj} = 0, r \not\equiv 0 \bmod n$.
- Если $r = 0 \bmod n$, то $\sum_{j=0}^{n-1} \alpha^{rj} = n$. Это не равно нулю, если n не кратно характеристике поля.

$$\frac{1}{n}\sum_{j=0}^{n-1} F_j \alpha^{-jk} = \frac{1}{n}\sum_{j=0}^{n-1} \alpha^{-jk} \sum_{l=0}^{n-1} \alpha^{lj} f_l = \frac{1}{n}\sum_{l=0}^{n-1} f_l \sum_{j=0}^{n-1} \alpha^{(l-k)j} = f_k$$

Существование дискретного преобразования Фурье

- Комплексные числа: существует для любых n
- Конечные поля $GF(p^m)$
  - n не может быть кратным характеристике поля p
  - α — образующий элемент циклической мультипликативной группы порядка n. Порядок $(p^m - 1)$ мультипликативной группы поля должен делиться на n (теорема Лагранжа)

Теорема о свертке

Теорема

Пусть существует ДПФ длины n *и* $e_i = f_i g_i, i = 0..n-1$. *Тогда*

$$E_j = \frac{1}{n}\sum_{k=0}^{n-1} F_{((j-k))} G_k.$$

Доказательство.

Вычислим ДПФ вектора с компонентами $e_i = f_i g_i$:

$$E_j = \sum_{i=0}^{n-1} \alpha^{ij} f_i g_i = \sum_{i=0}^{n-1} \alpha^{ij} f_i \frac{1}{n}\sum_{k=0}^{n-1} \alpha^{-ik} G_k = \frac{1}{n}\sum_{k=0}^{n-1} G_k \left(\sum_{i=0}^{n-1} \alpha^{i(j-k)} f_i\right) = \frac{1}{n}\sum_{k=0}^{n-1} G_k F_{((j-k))}$$

□

Алгоритмы быстрого преобразования Фурье могут быть использованы для вычисления свертки

Применение дискретного преобразования Фурье

- Обработка сигналов
- Сжатие изображений (JPEG: дискретное косинусное преобразование)
- Передача информации при наличии межсимвольной интерференции
- Быстрое умножение
- ...

Алгоритм Кули-Тьюки быстрого преобразования Фурье

- Рассмотрим вычисление ДПФ длины $n = n'n''$
- Пусть $i = i' + n'i'', k = n''k' + k''$. Тогда

$$F_k = F_{n''k'+k''} = \sum_{i=0}^{n-1} f_i \alpha^{ki} = \sum_{i''=0}^{n''-1} \sum_{i'=0}^{n'-1} \alpha^{(n''k'+k'')(i'+n'i'')} f_{i'+n'i''}, k' = 0..n'-1, k'' = 0..n''-1$$

- Пусть $\gamma = \alpha^{n'}, \beta = \alpha^{n''}$. Заметим, что $\forall k', i'' : \alpha^{n'n''k'i''} = 1$. Пусть $f_{i',i''} = f_{i'+n'i''}, F_{k',k''} = F_{n''k'+k''}$. Тогда

$$F_{k',k''} = \sum_{i'=0}^{n'-1} \beta^{k'i'} \left(\alpha^{i'k''} \sum_{i''=0}^{n''-1} \gamma^{i''k''} f_{i',i''} \right).$$

- Выражение во внутренних скобках зависит только от i', k'' и представляет собой набор ДПФ с ядром γ, результат которых после нескольких дополнительных умножений передается на вход другим ДПФ с ядром β

Алгоритм Кули-Тьюки быстрого преобразования Фурье

1. Вычислить $X_{i',k''} = \sum_{i''=0}^{n''-1} \gamma^{i''k''} f_{i',i''}, i' = 0..n'-1, k'' = 0..n''-1$. Непосредственная реализация требует $n'n''n''$ умножений и $n'n''(n''-1)$ сложений
2. $Y_{i',k''} = X_{i',k''}\alpha^{i'k''}, i' = 0..n'-1, k'' = 0..n''-1$. Это требует $n'n''$ умножений
3. Вычислить $F_{k',k''} = \sum_{i'=0}^{n'-1} \beta^{k'i'} Y_{i'k''}, k' = 0..n'-1$. Непосредственная реализация требует $n'n''n'$ умножений и $n'n''(n'-1)$ сложений.

- Общая сложность вычислений составляет $M(n) = n(n''+n'+1)$ умножений и $A(n) = n(n''+n'-2)$ сложений
- Иногда величины $\beta^{k'i'}$ имеют специальный вид, облегчающий вычисления $(-1, \mathrm{i}$ и т.п.)
- Дальнейшее снижение сложности возможно за счет рекурсивного использования алгоритмов БПФ малой длины. Тогда $M(n) = n'M(n'') + n''M(n') + n, A(n) = n'A(n'') + n''A(n')$

Вычисление ДПФ длины 2^m

- Прореживание по времени: $n' = 2, n'' = 2^{m-1}$, $\beta = \alpha^{n/2} = -1$ и $\gamma = \alpha^2$, т.е.

$$\begin{aligned} F_k &= \sum_{i=0}^{n/2-1} \gamma^{ik} f_{2i} + \alpha^k \sum_{i=0}^{n/2-1} \gamma^{ik} f_{2i+1} \\ F_{k+n/2} &= \sum_{i=0}^{n/2-1} \gamma^{ik} f_{2i} - \alpha^k \sum_{i=0}^{n/2-1} \gamma^{ik} f_{2i+1}, k = 0..n/2-1 \end{aligned}$$

- Прореживание по частоте: $n' = 2^{m-1}, n'' = 2$, $\gamma = \alpha^{n'} = \alpha^{n/2} = -1, \beta = \alpha^2$, т.е.

$$\begin{aligned} F_{2k'} &= \sum_{i'=0}^{n/2-1} \beta^{k'i'} \left(f_{i'} + f_{n/2+i'}\right) \\ F_{2k'+1} &= \sum_{i'=0}^{n/2-1} \beta^{k'i'} \left(\alpha^{i'}(f_{i'} - f_{n/2+i'})\right), k' = 0..n/2-1 \end{aligned}$$

Алгоритм Кули-Тьюки с прореживанием по частоте

Algorithm 1: DIFCooleyTukey(f,n,w)

```
L=1; N=n;
while N > 1 do
    N' = N/2;
    for (K = 0; K < L; K++) do
        J_w = 0;
        for
          (J = K·N; J < K·N + N'; J++)
          do
            W = w[J_w];
            T = f[J];
            f[J] = T + f[J + N'];
            f[J + N'] = W·(T − f[J + N']);
            J_w = J_w + L
    L = 2·L; N = N'
```

- $n = 2^m$
- Массив w содержит $\alpha^i, 0 \leq i < n/2$
- Пусть $k = \sum_{i=0}^{m-1} k_i 2^i$, $k_i \in \{0, 1\}$. Обозначим элементы массивов их двоичными индексами, выписанными в порядке от наименее значащего бита к старшему биту, т.е. $A_k = A_{k_0, k_1, \ldots, k_{l-1}}$
- После завершения работы вектор f содержит последовательность $F_{k_{l-1}, k_{l-2}, \ldots, k_0}$, $k = 0..n-1$
- $\frac{1}{2} n \log n$ умножений, $n \log n$ сложений
- Сложность может быть снижена за счет упрощенной обработки $W \in \{\pm 1, \pm \mathrm{i}\}$

Представление целых чисел в ЭВМ

- Система счисления по основанию b

$$a = \sum_{i \geq 0} a_i b^i, 0 \leq a_i < b$$

- Отрицательные числа
  1. Прямой код или абсолютное значение со знаком, например -1234
     Недостаток: существование +0 и -0, т.е. двух кодов, обозначающих 0
  2. Дополнительный код: $-a, a > 0$ представляется как $b^n - a$, где n — используемая разрядность. Это эквивалентно вычислению по модулю b^n
     $n = 10, b = 10 : -1 = (99999\,99999)_{10}$; $n = 16, b = 2$: $-1 = (1111\,1111\,1111\,1111)_2$.
     Недостаток: несимметричность относительно нуля: для числа $-2^{n-1} = (100\ldots0)_2$ невозможно представить обратное.
  3. Обратный код: каждый разряд a_i модуля отрицательного числа a заменяется на $b - 1 - a_i$. Это эквивалентно вычислениям по модулю $b^n - 1$
     Пример: $-1 = (99999\,99998)_{10}$
     Недостаток: наличие двух представлений нуля

Сложение и умножение в столбик

Algorithm 2: Add(u,v,n)

$j = 0$; $k = 0$;
while $j<n$ **do**
  $w_j = u_j + v_j + k$;
  if $w_j \geq b$ **then**
    $w_j = w_j - b$;
    $k = 1$
  else
    k=0
  j=j+1
$w_n = k$;
return $(w_n, \ldots, w_0)$

Сложность $\Theta(n)$

Algorithm 3: Multiply(u,v,m,n)

$w_j = 0, j = 0..m-1; j = 0$;
while $j<n$ **do**
  if $v_j > 0$ **then**
    i=0;k=0;
    while $i < m$ **do**
      $t = u_i v_j + w_{i+j} + k$; $w_{i+j} = t \bmod b; k = \lfloor t/b \rfloor$; $i = i+1$
    $w_{j+m} = k$
  else
    $w_{j+m} = 0$
  j=j+1
return $(w_{m+n-1}, \ldots, w_0)$

Сложность $\Theta(n^2)$

Быстрое умножение

- Всякое натуральное число $A=\sum_{i=0}^{n-1} a_i b^i, 0 \leq a_i < n$ можно представить как $A=a(b), a=\sum_{i=0}^{n-1} a_i x^i$
- $A \cdot B = c(b)$, где $c(x)=a(x)b(x)$
- Применим любой быстрый алгоритм линейной свертки
- Осторожно: коэффициенты $c(x)$ могут быть $\geq b$. Необходим учет переносов

Расширенный алгоритм Евклида

Поиск наибольшего общего делителя $r_{-1} = a, r_0 = b$

$$r_{i-1} = q_i r_i + r_{i+1}, r_{i+1} < r_i$$

НОД равен последнему ненулевому остатку r_i

$$\begin{pmatrix} r_i & r_{i-1} \end{pmatrix} \begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} r_{i+1} & r_i \end{pmatrix}$$

$$\begin{pmatrix} b & a \end{pmatrix} \underbrace{\prod_i \begin{pmatrix} -q_i & 1 \\ 1 & 0 \end{pmatrix}}_{U} = \begin{pmatrix} 0 & \gcd(a, b) \end{pmatrix}$$

Теорема (Безу)

Существуют $u, v : bu + av = \gcd(a, b)$

Нахождение мультипликативно обратного элемента

- Пусть даны $a, b \in \mathbb{N}$. Найти $v : av \equiv 1 \bmod b$

Нахождение мультипликативно обратного элемента

- Пусть даны $a, b \in \mathbb{N}$. Найти $v : av \equiv 1 \bmod b$
- $bu + av = \gcd(a, b) \Rightarrow av \equiv \gcd(a, b) \bmod b$
- Если $\gcd(a, b) = 1$, то $v \equiv a^{-1} \bmod b$ существует и может быть найдено с помощью расширенного алгоритма Евклида

Метод Монтгомери модулярного умножения: $c = ab \bmod N$

- Выберем $R > N, (R, N) = 1$. Удобно $R = 2^k$
- Числа в форме Монтгомери: $a' \equiv aR \bmod N$, $b' \equiv bR \bmod N$
- $c' \equiv cR = (a'b')\tilde{R} \bmod N$, где $R\tilde{R} = qN + 1$ для некоторого q, т.е. $\tilde{R} \equiv R^{-1} \bmod N$
- Нахождение $T\tilde{R} \bmod N$:
  1. $m = (q(T \bmod R)) \bmod R$
  2. $t = (T + mN)/R$
  3. Если $t \geq N$, возвратить $t - N$, иначе t
- Заметим, что $mN \equiv TqN \bmod R$. При этом $qN + 1 \equiv 0 \bmod R$, т.е. $Tqn \equiv -T \bmod R$
- Следовательно, $T + mn \equiv 0 \bmod R$, т.е. на втором шаге деление выполняется нацело
- $tR = T + mN \equiv T \bmod N$, т.е $t \equiv TR^{-1} \bmod N$. Если $0 \leq T < RN$, то получим $t < 2N$. Таким образом, результат шага 3 всегда меньше чем N

Метод Монтгомери модулярного умножения

Algorithm 4: MM(a', b', N)

```
T = a'b';
m ≡ qT mod R;
t = (T + mN)/R;
if t ≥ N then
    return t-N
else
    return t
```

Результат: $c' \equiv (a'b')R^{-1} \bmod N$

- $MM(a', b', N)$ требует 3 умножений
- При $R = 2^k$ операции $\bmod R$ и $/R$ тривиальны
- Преобразование a, b в форму Монтгомери:

$$a' = MM(a, R^2, N), b' = MM(b, R^2, N)$$

- Обратное преобразование: $c = MM(c', 1, N)$
- Вычисление $c = ab \bmod N$:
  1. $a' = MM(a, R^2, N)$
  2. $b' = MM(b, R^2, N)$
  3. $c' = MM(a', b', N)$
  4. $c = MM(c', 1, N)$

Быстрое возведение в степень

- Рассмотрим вычисление $y = x^m, m \in \mathbb{N}$

Быстрое возведение в степень

- Рассмотрим вычисление $y = x^m, m \in \mathbb{N}$
- $m = \sum_{i=0}^{t} m_i 2^i, m_i \in \{0, 1\}$
- Двоичный метод возведения в степень

$$y = x^{\sum_{i=0}^{t} m_i 2^i} = \prod_{i=0}^{t} (x^{2^i})^{m_i}$$

- Сложность $t + \sum_{i=0}^{t} m_i - 1$ умножений
- Алгоритм не является оптимальным
  - $y = x^{15} = (x^3)^5$: $p_1 = x^3 = x \cdot x \cdot x$; $p_2 = p_1^2$; $p_3 = p_2^2$; $y = p_3 p_1$ — 5 умножений
  - Двоичный метод: 6 умножений

Безопасная передача данных по открытым каналам

- Как Алисе безопасно передать Бобу по открытому каналу сообщение m?

Безопасная передача данных по открытым каналам

- Как Алисе безопасно передать Бобу по открытому каналу сообщение m?
- Боб: открытый (публичный) ключ P, закрытый (секретный) ключ S
- Алиса отправляет $c = f(P, m)$ по открытому каналу
- Боб: $m = g(S, c)$
- Требования:
    - $g(S, f(P, m)) = m$
    - Сложно найти m, зная P и c
    - Сложно найти S, зная P

Криптосистема Ривеста-Шамира-Адельмана (RSA)

Теорема (Эйлера)

Если a и m взаимно просты, то $a^{\phi(m)} \equiv 1 \mod m$, где $\phi(m)$ — количество натуральных чисел $i : 1 \leq i \leq m$, взаимно простых с m (функция Эйлера)

$$\phi\left(\prod_i p_i^{a_i}\right) = \prod_i \left(p_i^{a_i} - p_i^{a_i-1}\right), p_i - \text{ простые числа}$$

- Выберем два больших различных простых числа p, q, зафиксируем число $e \in \mathbb{N}$
- $m = pq; \phi(m) = (p-1)(q-1)$. Найдем $d : ed \equiv 1 \mod \phi(m)$, т.е. $ed = 1 + s\phi(m)$
- Открытый ключ: $P = [m, e]$. Секретный ключ: $S = [m, d]$
- Шифрование сообщения x: $y = f(P, x) \equiv x^e \mod m$
- Дешифрование криптограммы y: $x = g(S, y) \equiv y^d \equiv x^{ed} \equiv x^{1+s\phi(m)} \equiv x \mod m$

Безопасность RSA

- Разложение целого числа z на множители
  - Классические компьютеры: метод обобщенного решета числового поля со сложностью $O(e^{\alpha(\log^{1/3}(z))\log^{2/3}\log z})$
  - Квантовые компьютеры: алгоритм Шора со сложностью $O(\log z)$, требует $O((\log^2 z)(\log\log z)\log\log\log z)$ квантовых вентилей
    Продемонстрировано разложение числа $4088459 \approx 2^{22}$ на 5-кубитном квантовом компьютере
- Используются числа длиной $\log z \approx 1024$–4096 бит

Проверка чисел на простоту

- Метод пробного деления: попытаться поделить проверяемое m на различные $b < \sqrt{m}$
- Теорема Ферма: для любого простого m и всех $b < m$ справедливо сравнение $b^{m-1} \equiv 1 \bmod m$
- Если это сравнение не выполняется, число m является составным
- Псевдопростое число m для основания b: $b^{m-1} \equiv 1 \bmod m$
- Числа Кармайкла (абсолютно псевдопростые) — псевдопростые для любого основания
  - Достаточно наличия у $m = \prod p_i$ трех или более нечетных простых делителей $p_i : (p_i - 1)|(m-1)$
    По теореме Ферма $b^{p_i-1} \equiv 1 \bmod p_i$. Т.к. $(p_i - 1)|(m-1)$, справедливо $b^{m-1} \equiv 1 \bmod p_i$. В силу китайской теоремы об остатках $b^{m-1} \equiv 1 \bmod m$.
  - Наименьшим таким числом m является $561 = 3 \cdot 11 \cdot 17$.

Вероятностный тест Рабина-Миллера

- Пусть m — тестируемое число
- Выберем $b : (b, m) = 1$. Если окажется, что $(b, m) > 1$, m — составное
- Пусть $m - 1 = t2^s$, где t — нечетное число. Рассмотрим числа $x_r \equiv b^{t2^r} \bmod m, 0 \leq r < s,$, причем будем выбирать наименьший по абсолютной величине вычет x_r
- В силу теоремы Ферма, если число m простое, то $b^{m-1} \equiv 1 \bmod m$
- Извлечем квадратный корень из обеих частей этого сравнения, т.е. исследуем $x_{s-1} = b^{(m-1)/2} \bmod m$. При простом m множество решений сравнения $x^2 \equiv 1 \bmod m$ исчерпывается ± 1, т.е. x_{s-1} может быть сравнимо либо с 1, либо с -1
  - Если $x_{s-1} \equiv -1 \bmod m$, откажемся от принятия решения
  - Если $x_{s-1} \equiv 1 \bmod m$, рассмотрим $x_{s-2}, \ldots, x_0$
  - В противном случае m — составное число

Вероятностный тест Рабина-Миллера

1. Выбрать целое число $b : 1 < b < m$.
2. Если $(b, m) \neq 1$, возвратить "m — составное".
3. Найти числа $t, s : m - 1 = t2^s$, где t — нечетное число.
4. Если $b^t \not\equiv 1 \bmod m \wedge \forall r : 0 \leq r \leq s - 1 : b^{2^r t} \not\equiv -1 \bmod m$, возвратить "$m$ — составное".
5. Иначе возвратить "не удалось определить".

- Если алгоритм возвратил "не удалось определить", вероятность того, что число все-таки окажется в этом случае составным, не превосходит $1/4$
- Алгоритм следует запускать многократно с различными b до получения приемлемой вероятности того, что число составное
- Тест может быть скомбинирован с методом пробного деления

Быстрое умножение матриц

- $P = \underbrace{Q}_{m \times n} \underbrace{S}_{n \times r} : P_{ij} = \sum_{t=0}^{n-1} Q_{it} S_{tj}, 0 \le i < m, 0 \le j < r$
  Сложность mnr умножений, $mr(n-1)$ сложений
- Алгоритм Штрассена

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} A & C \\ B & D \end{pmatrix} = \begin{pmatrix} aA + bB & w + v + (b - (c - a + d))D \\ w + u + d(B - (A + D - C)) & w + u + v \end{pmatrix},$$

$$u = (c - a)(C - D), v = (c + d)(C - A), w = aA + (c - a + d)(A + D - C),$$

  Сложность 7 умножений и 15 сложений
- Рекурсивное применение алгоритма Штрассена для $n \times n$ матриц: $T(n) = 7T(n/2) + 15(n/2)^2 = \Theta(n^{2.8074})$.

Умножение двоичных матриц

- $P = QS$, где Q, S — $n \times n$ матрицы над полукольцом $\mathcal{B} = (\{0, 1\}, \vee, \wedge)$
- Алгоритм Штрассена
  - Непосредственное применение невозможно по причине отсутствия вычитания
  - Погрузить $\mathcal{B}$ в кольцо целых чисел, заменить в P все ненулевые элементы на 1
- Алгоритм 4 русских (Кронрод, Арлазаров, Диниц, Фараджев)
  - $Q = (Q_1 | Q_2 | \dots | Q_{n/\log_2 n})$, $S = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_{n/\log_2 n} \end{pmatrix}$. Тогда $QS = \bigvee_{i=1}^{n/\log_2 n} Q_i S_i$
  - Каждая строка Q_i содержит $\log_2 n$ элементов из $\{0, 1\}$ и задает дизъюнкцию некоторого подмножества строк S_i
  - Перебирать подмножества так, чтобы очередное подмножество отличалось от *какого-либо из предыдущих добавлением одного* элемента.
  - произведение $Q_i S_i$ может быть вычислено не более чем за n^2 операций
  - сложность $\Theta(n^3 / \log_2 n)$ операций.

Транспонирование матриц: кэш-независимый алгоритм

- Пусть дана $m \times n$ матрица A
- Транспонированной матрицей A^T называется матрица B с элементами $B_{ij} = A_{ji}, i = 1..n, j = 1..m$
- Пусть результат этой операции записывается в область памяти, не пересекающуюся с областью памяти, занимаемой исходной матрицей A
- Кеширование: время выполнения операций перемещения данных существенно зависит от того, в каком порядке они выполняются
- Предположим, что имеется кеш-память общим размером Z с размером кеш-линий L
- Если $m \gg Z/L$ и $n \gg Z/L$, то реализация транспонирования через два вложенных цикла приводит к кеш-промаху при каждой операции записи/чтения, т.е. число кеш-промахов $O(mn)$
- RECTRANSPOSE: Разобьем матрицу A по наибольшей размерности (пусть $n \geq m$) на две подматрицы. Тогда $A = (A_1|A_2)$ и $B = A^T = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$, где $B_1 = A_1^T$ и $B_2 = A_2^T$

Транспонирование матриц

Теорема

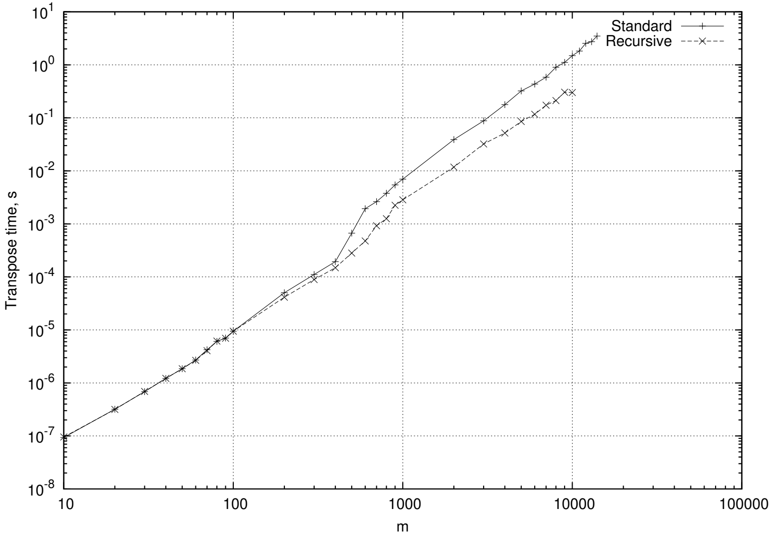
При выполнении процедуры RECTRANSPOSE происходит $O(1 + mn/L)$ кеш-промахов.

Доказательство.

Пусть α — величина, такая что две матрицы размером $M \times N$ и $N \times M$, $\max\{M, N\} \leq \alpha L$, полностью умещаются в кеш-памяти

1. $\max\{m, n\} \leq \alpha L$: матрицы A и B целиком помещаются в кеш-памяти и занимают $O(1) + 2mn/L$ кеш-линий. Число кеш-промахов $O(1 + mn/L)$.
2. $m \leq \alpha L < n$. После нескольких шагов разбиения задача будет сведена к задачам транспонирования матриц размерностью $m \times n'$, $\alpha L/2 \leq n' \leq \alpha L$. Каждая из них приводит к $O(1 + mn'/L)$ кеш-промахам, а общее число кеш-промахов составит $O(1 + mn/L)$. Случай $n \leq \alpha L < m$ рассматривается аналогично.
3. Случай $m, n > \alpha L$ также рекурсивно сводится к задачам малой размерности, т.е. число кеш-промахов $O(1 + mn/L)$.

Транспонирование $m \times m$ матриц



Заключение

- Линейная и циклическая свертка: алгоритм Карацубы, связь с дискретным преобразованием Фурье
- Быстрое преобразование Фурье: алгоритм Кули-Тьюки
- Умножение целых чисел как свертка
- Метод модулярного умножения Монтгомери
- Двоичный метод возведения в степень
- Криптосистема RSA
- Алгоритм Штрассена умножения матриц
- Кэш-независимые алгоритмы