Алгоритмы и структуры данных

Лекция 2. Теоретико-числовые алгоритмы.

Артур Кулапин

ВШПИ МФТИ

16 сентября 2023 г.

О чем разговор?

1. Функция Эйлера
   - Факторизация
   - Функция Эйлера
2. Решето Эратосфена
   - Классическое решето
   - Линейное по времени решето
3. Модульная арифметика
   - Кольца
   - Деление
4. Алгоритм Евклида
   - Классика

Факторизация

Def. Факторизация — процесс разложения числа на делители.

Факторизация

Def. Факторизация — процесс разложения числа на делители.

1. Линейный алгоритм. Для каждого i от 1 до n проверяем, что число n делится на i.

Факторизация

Def. Факторизация — процесс разложения числа на делители.

1. Линейный алгоритм. Для каждого i от 1 до n проверяем, что число n делится на i.
2. Вспомним, что если d — делитель n, то и n/d — делитель числа n, а значит достаточно перебирать не до n, а до $[\sqrt{n}]$

Функция Эйлера

Def. Функция Эйлера от числа n, $\varphi(n)$, равна количеству чисел, меньших n и взаимно простых с ним.

Функция Эйлера

Def. Функция Эйлера от числа n, $\varphi(n)$, равна количеству чисел, меньших n и взаимно простых с ним.

Например, $\varphi(5) = 4$ (числа $1, 2, 3, 4$).

Функция Эйлера

Def. Функция Эйлера от числа n, $\varphi(n)$, равна количеству чисел, меньших n и взаимно простых с ним.

Например, $\varphi(5) = 4$ (числа $1, 2, 3, 4$).

Свойства функции Эйлера.

1. Если p — простое, то $\varphi(p) = p - 1$.

Функция Эйлера

Def. Функция Эйлера от числа n, $\varphi(n)$, равна количеству чисел, меньших n и взаимно простых с ним.

Например, $\varphi(5) = 4$ (числа $1, 2, 3, 4$).

Свойства функции Эйлера.

1. Если p — простое, то $\varphi(p) = p - 1$.
2. Если p — простое, то $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$.

Функция Эйлера

Def. Функция Эйлера от числа n, $\varphi(n)$, равна количеству чисел, меньших n и взаимно простых с ним.

Например, $\varphi(5) = 4$ (числа $1, 2, 3, 4$).

Свойства функции Эйлера.

1. Если p — простое, то $\varphi(p) = p - 1$.
2. Если p — простое, то $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$.
3. Если n, m взаимно просты, то $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.

Вычисление $\varphi(n)$

Наивный алгоритм. Для каждого от 1 до n проверяем, является ли делителем n.

Вычисление $\varphi(n)$

Наивный алгоритм. Для каждого от 1 до n проверяем, является ли делителем n.

Немного преобразований.

Вычисление $\varphi(n)$

Наивный алгоритм. Для каждого от 1 до n проверяем, является ли делителем n.

Немного преобразований.

$$\varphi(n) = \varphi\left(\prod_i p_i^{\alpha_i}\right) = \prod_i \left(\varphi(p_i^{\alpha_i})\right) = \prod_i \left(p_i^{\alpha_i} - p_i^{\alpha_i - 1}\right) =$$
$$= \prod_i \left(p_i^{\alpha_i}\left(1 - \frac{1}{p_i}\right)\right) = \prod_i p_i^{\alpha_i} \cdot \prod_i \left(1 - \frac{1}{p_i}\right) = n \cdot \prod_i \left(1 - \frac{1}{p_i}\right)$$

Вычисление $\varphi(n)$

Наивный алгоритм. Для каждого от 1 до n проверяем, является ли делителем n.

Немного преобразований.

$$\varphi(n) = \varphi\left(\prod_i p_i^{\alpha_i}\right) = \prod_i \left(\varphi(p_i^{\alpha_i})\right) = \prod_i \left(p_i^{\alpha_i} - p_i^{\alpha_i - 1}\right) =$$
$$= \prod_i \left(p_i^{\alpha_i}\left(1 - \frac{1}{p_i}\right)\right) = \prod_i p_i^{\alpha_i} \cdot \prod_i \left(1 - \frac{1}{p_i}\right) = n \cdot \prod_i \left(1 - \frac{1}{p_i}\right)$$

Таким образом, достаточно знать факторизацию числа, а из нее можно вычислить $\varphi(n)$ за $\mathcal{O}(\log n)$.

Решето Эратосфена

Def. Решето Эратосфена — алгоритм, позволяющий найти все простые числа от 2 до n.

Algorithm.

1. Берем минимальное непросмотренное число k, объявляем его простым.
2. Далее все числа от 2 до n, кратные k объявляются непростыми.
3. Повторить, пока все числа не рассмотрены.

Псевдокод

```
bool primes[n];
primes[0] = primes[1] = false;
for (int i = 2; i < n; ++i) {
  primes[i] = true;
  for (int j = 2; i * j < n; ++j) {
    primes[i * j] = false;
  }
}
```

Псевдокод

```
bool primes[n];
primes[0] = primes[1] = false;
for (int i = 2; i < n; ++i) {
  primes[i] = true;
  for (int j = 2; i * j < n; ++j) {
    primes[i * j] = false;
  }
}
```

Какие проблемы вы видите в коде? Какие есть точки роста?

Оптимизируем

Оптимизируем

1. Самое простое — можно увеличивать i на 2 каждый раз, так как кроме двойки все простые нечетные.

Оптимизируем

1. Самое простое — можно увеличивать i на 2 каждый раз, так как кроме двойки все простые нечетные.
2. Можно начинать перебирать j начиная с i, поскольку все числа меньшие i^2, кратные i, обязательно имеют простой делитель меньше i, а значит, все они уже были отсеяны раньше.

Улучшенный псевдокод

```
bool primes[n]{false};
primes[0] = primes[1] = false;
primes[2] = true;
for (int i = 3; i < n; i += 2) {
  primes[i] = true;
  for (int j = i; i * j < n; ++j) {
    primes[i * j] = false;
  }
}
```

Сложность решета Эратосфена

Теорема (б/д). Сложность алгоритма выше составляет $\mathcal{O}(n \log \log n)$, где n — число, до которого мы ищем простые.

Линейный алгоритм

Решим другую задачу. Для каждого числа от 2 до n найдем минимальный простой делитель (массив `mind`). Тогда простые легко детектируются.

При рассмотрении очередного числа i пройдемся по массиву уже найденных простых `primes` и будем обновлять минимальный делитель у просматриваемых чисел.

Псевдокод

```
Array<int> primes;
Array<int> mind(n + 1);
for (int i = 2; i <= n; ++i) { mind[i] = i; }
for (int i = 2; i <= n; ++i) {
    if (mind[i] == i) { primes.push_back(i); }
    for (int p : primes) {
        if (p * i > n || p > mind[i]) {
            break;
        }
        mind[i * p] = p;
    }
}
```

Сложность

Заметим, что сложность определяется суммарным числом итераций внутреннего `for`.

Сложность

Заметим, что сложность определяется суммарным числом итераций внутреннего `for`.

Сколько раз для конкретного k изменяется `mind[k]`?

Сложность

Заметим, что сложность определяется суммарным числом итераций внутреннего `for`.

Сколько раз для конкретного k изменяется `mind[k]`?

Казалось бы, для каждого простого p в разложении k изменяется `mind[k]`. Однако можно заметить, что на самом деле это не так. Рассмотрев `mind[k]` один раз, больше он не будет обновлен.

Сложность

Заметим, что сложность определяется суммарным числом итераций внутреннего `for`.

Сколько раз для конкретного k изменяется `mind[k]`?

Казалось бы, для каждого простого p в разложении k изменяется `mind[k]`. Однако можно заметить, что на самом деле это не так. Рассмотрев `mind[k]` один раз, больше он не будет обновлен.

Откуда следует, что сложность линейная.

Сложность

Заметим, что сложность определяется суммарным числом итераций внутреннего `for`.

Сколько раз для конкретного k изменяется `mind[k]`?

Казалось бы, для каждого простого p в разложении k изменяется `mind[k]`. Однако можно заметить, что на самом деле это не так. Рассмотрев `mind[k]` один раз, больше он не будет обновлен.

Откуда следует, что сложность линейная.

Бонус! Можно получать факторизацию любого числа k от 1 до n за $\mathcal{O}(\log k)$ с предподсчетом за $\mathcal{O}(n)$.

Зачем нужно что-либо кроме $\mathbb{R}$ или $\mathbb{Z}$?

Иногда возникает нужда выйти за рамки привычных нам множеств, например, целых или вещественных чисел.

Были созданы математические абстракции для работы с нестандартными множествами «чисел». Одной из таких является *кольцо*.

Def. Кольцо — набор из множества «чисел» и операций сложения и умножения, для которых верны все *классические* свойства этих операций: ассоциативность, коммутативность, дистрибутивность... Однако нет обратимости по умножению (нельзя делить).

Примечание. Это наивное определение, которое в общем некорректно, но дает понимание того, с чем мы работаем.

Кольца

Credits

Кольца $\mathbb{Z}_m$

Def. Кольцо $\mathbb{Z}_m$ — структура над числами $0, 1, \ldots, m-1$, где все операции ведутся по модулю m.

Пример. Рассмотрим кольцо $\mathbb{Z}_6$, то есть числа $0, 1, 2, 3, 4, 5$. И посчитаем выражение $5 \cdot 4 + 3 \cdot 2$ в этом кольце.

$5 \cdot 4 = 20 \equiv 2 \pmod 6 \quad 3 \cdot 2 = 6 \equiv 0 \pmod 6$, откуда итоговое значение сравнимо с 2 по модулю 6 или 2 в $\mathbb{Z}_6$.

Обратимость в $\mathbb{Z}_m$

Как было сказано в определении, есть сложение и умножение. А что с обратными операциями?

Обратимость в $\mathbb{Z}_m$

Как было сказано в определении, есть сложение и умножение. А что с обратными операциями?

Def. Отрицательное число в $\mathbb{Z}_m$.

$$-b = \begin{cases} 0, & b = 0 \\ m - b, & \text{иначе} \end{cases}$$

Таким образом, $2 - 4$ в $\mathbb{Z}_6$ следует интерпретировать как

$$2 + (-4) = 2 + (6 - 4) = 2 + 2 = 4$$

Деление в $\mathbb{Z}_m$

Def. $a^{-1} = b$ в $\mathbb{Z}_m$ тогда и только тогда, когда $a \cdot b \equiv 1 \pmod{m}$.

Деление в $\mathbb{Z}_m$

Def. $a^{-1} = b$ в $\mathbb{Z}_m$ тогда и только тогда, когда $a \cdot b \equiv 1 \pmod{m}$.

Пример. Рассмотрим $\mathbb{Z}_5$ и найдем 3^{-1} в нем. Нетрудно заметить, что $2 = 3^{-1}$ в $\mathbb{Z}_5$.

Пример. Рассмотрим $\mathbb{Z}_6$ и найдем 3^{-1} в нем.

Деление в $\mathbb{Z}_m$

Def. $a^{-1} = b$ в $\mathbb{Z}_m$ тогда и только тогда, когда $a \cdot b \equiv 1 \pmod{m}$.

Пример. Рассмотрим $\mathbb{Z}_5$ и найдем 3^{-1} в нем. Нетрудно заметить, что $2 = 3^{-1}$ в $\mathbb{Z}_5$.

Пример. Рассмотрим $\mathbb{Z}_6$ и найдем 3^{-1} в нем. Какое число k не было бы взято в качестве 3^{-1}, $3 \cdot k \pmod{6} \in 0, 3$, то есть не равно единице.

Поля

Def. Поле — набор из множества «чисел» и операций сложения и умножения, для которых верны все *классические* свойства этих операций: ассоциативность, коммутативность, дистрибутивность и обратимость.

Примечание. Это наивное определение, которое в общем некорректно, но дает понимание того, с чем мы работаем.

Матчасть

Теорема (Малая теорема Ферма) (б/д). Пусть m — простое, тогда $a^{-1} = a^{m-2}$ в $\mathbb{Z}_m$.

Следствие. Если p — простое, то $\mathbb{Z}_p$ — поле, обратное тоже верно. То есть для каждого ненулевого $a \in \mathbb{Z}_p$ существует a^{-1}.

Матчасть

Теорема (Малая теорема Ферма) (б/д). Пусть m — простое, тогда $a^{-1} = a^{m-2}$ в $\mathbb{Z}_m$.

Следствие. Если p — простое, то $\mathbb{Z}_p$ — поле, обратное тоже верно. То есть для каждого ненулевого $a \in \mathbb{Z}_p$ существует a^{-1}.

Теорема (Эйлер) (б/д). Пусть $m \in \mathbb{N}$. Тогда для $a \in \mathbb{Z}_m$ существует a^{-1} тогда и только тогда, когда $gcd(a, m) = 1$. Более того, если $gcd(a, m) = 1$, то $a^{-1} = a^{\varphi(m)-1}$.

Таким образом, деление, сводящееся к поиску обратного, выполнимо в условиях теорем выше.

Алгоритм Евклида

```
int gcd (int a, int b) {
    while (b != 0) {
        a %= b;
        swap(a, b);
    }
    return a;
}
```

Алгоритм Евклида

```
int gcd (int a, int b) {
    while (b != 0) {
        a %= b;
        swap(a, b);
    }
    return a;
}

int GCD(int a, int b) {
    return b ? GCD(b, a % b) : a;
}
```

Алгоритм Евклида

```
int gcd (int a, int b) {
    while (b != 0) {
        a %= b;
        swap(a, b);
    }
    return a;
}

int GCD(int a, int b) {
    return b ? GCD(b, a % b) : a;
}

std::gcd(n, m); // C++
```

Расширенный алгоритм Евклида

Рассмотрим уравнение $ax + by = gcd(a, b)$. Пусть было найдено решение для $(b\%a, a)$, перейдем к (a, b).

$b\%a = b - \lfloor \frac{b}{a} \rfloor \cdot a$, подставим в уравнение:

$\left(b - \lfloor \frac{b}{a} \rfloor \cdot a\right) x_1 + ay_1 = gcd(a, b)$

Откуда $a\left(y_1 - \lfloor \frac{b}{a} \rfloor x_1\right) bx_1 = gcd(a, b)$

Следовательно

$$\begin{cases} x = y_1 - \lfloor \frac{b}{a} \rfloor x_1 \\ y = x_1 \end{cases}$$

В поисках обратного

Рассмотрим уравнение $ax + my = 1$. Возьмем обе части по модулю m и $ax = 1 \Longrightarrow x = a^{-1}$.

В поисках обратного

Рассмотрим уравнение $ax + my = 1$. Возьмем обе части по модулю m и $ax = 1 \Longrightarrow x = a^{-1}$.

То есть с помощью расширенного алгоритма Евклида ищем обратный элемент в $\mathbb{Z}_m$ за $\mathcal{O}(\log m)$.