Алгоритм Евклида

Сегодня мы поговорим об одном из самых древних алгоритмов – алгоритме Евклида. Он позволяет для двух натуральных чисел a и b найти наибольший общий делитель, то есть найти такое максимальное g такое, что и a, и b делятся нацело на g. В английском языке такое число называется *greatest common divisor (gcd)*.

1 Ключевой факт алгоритма

Алгоритм основывается на одном главном факте: для любых двух натуральных чисел a и b (без потери общности скажем, что $a \geq b$) верно следующее равенство:

$$\text{НОД}(a, b) = \text{НОД}(a - b, b)$$

Обоснуем этот факт. Обозначим за g значение НОД(a, b). Тогда, по определению, a кратно g и b кратно g. Отсюда следует, что $a - b$ тоже кратно g, а значит g – это общий делитель чисел $a - b$ и b, а значит не превосходит наибольшего из всех общих делителей $a - b$ и b:

$$g = \text{НОД}(a, b) \leq \text{НОД}(a - b, b)$$

Аналогично рассуждая, давайте обозначим $g' = \text{НОД}(a - b, b)$. Значит $a - b$ кратно g' и b кратно g'. Значит сумма $(a - b) + b = a$ кратна g'. Получается g' является одновременно делителем и a, и b, значит не превосходит наибольшего из общих делителей a и b:

$$g' = \text{НОД}(a - b, b) \leq \text{НОД}(a, b)$$

Так мы получили, что $g' \leq g$ и $g \leq g'$ по первой части рассуждения, а значит $g = g'$. Мы доказали наш главный факт: НОД$(a, b) =$ НОД$(a - b, b)$.

2 Первая версия алгоритма

Отметим тот факт, что 0 кратен любому числу, поэтому если одно из чисел 0, то мы нашли НОД – он равен второму значению. Это приводит нас к первой версии алгоритма. Напишем рекурсивную функцию `gcd(a, b)`, которая возвращает наибольший общий делитель чисел `a` и `b`. Давайте для удобства будем считать, что `a >= b` (если это не так, вернем значение функции, поменяв местами аргументы):

```
def gcd(a, b):
    if a < b:
        return gcd(b, a)
```

У нашей рекурсии должна быть точка выхода. Если `b == 0`, то, как мы уже обсуждали, наибольший общий делитель равен `a` – вернем его. А если `b != 0`, вернем `gcd` от `a - b` и `b`:

```
def gcd(a, b):
    if a < b:
        return gcd(b, a)
    if b == 0:
        return a
    return gcd(a-b, b)
```

Алгоритм получился непростым и требующим долгих математических обоснований. Давайте промоделируем его на числах 18 и 15. Первый запуск, `a = 18` и `b = 15`, приведет к запуску с параметрами `3` и `15`, ведь `18 - 15 = 3`. Дальше будет обмен позициями аргументов, потому что `b` оказалось больше `a`. После этого, мы последовательно запустимся от аргументов `12` и `3`, `9` и `3`, `6` и `3`, `3` и `3`, и, наконец, `0` и `3`. После этого происходит снова обмен аргументами и выход из рекурсии, с возвратом значения `3` – наибольший общий делитель найден:

`a`	`b`
18	15
3	15
15	3
12	3
9	3
6	3
3	3
0	3
3	0

То есть, наш алгоритм нашел цепочку равенств НОД$(18, 15)$ = НОД$(3, 15)$ = ... = НОД$(3, 0) = 3$.

3 Оптимизация алгоритма

Теперь оптимизируем наш алгоритм. Когда мы моделировали процесс, то начиная со стадии `a = 15` и `b = 3`, мы долго вычитали `3` из `a`. А представьте, если бы `a` было равно миллиарду, а `b` – единице. Мы бы очень долго вычитали единицу и в конце концов превысили бы ограничение на глубину рекурсии. Попробуем решить эту проблему.

Давайте разделим a на b, то есть представим b в виде $a = kb + r$, где r – это остаток от деления, число от 0 до $b - 1$. Тогда согласно главному факту, выполняется следующее равенство:

$$\text{НОД}(a, b) = \text{НОД}(a-b, b) = \text{НОД}(a-2b, b) = ... = \text{НОД}(a-kb, b) = \text{НОД}(r, b) = \text{НОД}(b, r)$$

Другими словами, мы можем не проделывать промежуточные вычитания, а сразу перейти от a, b к паре r, b. Более того, мы точно знаем, что $r < b$, поэтому перейдем сразу к паре чисел b, r, чтобы поддерживать тот факт, что первое число в паре больше второго.

Модифицируем соответствующим образом наш код. Если `b != 0`, то мы можем сразу перейти к вызову функции от чисел `b` и `a % b`:

```
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)
```

Теперь в случае с 18 и 15 у нас будет такой порядок вызовов:

a	b
18	15
15	3
3	0

А, например, для миллиарда и единицы будет только вызов от единицы и ноля. Такая функция работает очень быстро – за время порядка логарифма максимального числа.

4 Нерекурсивный алгоритм Евклида

Теперь напишем и нерекурсивный алгоритм Евклида.

Если можно легко избавиться от рекурсии, то лучше это сделать, потому что рекурсия тратит время на то, чтобы вернуть ответ через все вызовы. Когда мы достигаем нижнего уровня рекурсии, то есть точки выхода, рекурсия начинает раскручиваться назад и возвращать на уровень выше ответ, тогда как подход с циклом этого не потребует.

Итак, мы решили, что нам удобно поддерживать `a >= b` – если это не так, обменяем значения переменных в самом начале командой `a, b = b, a`. Такая запись элегантно присваивает в `a` значение `b`, а в `b` значение `a` – это происходит одновременно:

```
def gcd(a, b):
    if a < b:
        a, b = b, a
```

Теперь, будем выполнять действия пока `b > 0`. Действие простое: нам нужно от пары `a`, `b` перейти к паре `b`, `a % b`. Давайте сделаем такое же присваивание: в `a` присвоим `b`, а в `b` присвоим `a % b`. После завершения цикла, ответ лежит в переменной `a` – вернем его:

```
def gcd(a, b):
    if a < b:
        a, b = b, a
    while b > 0:
        a, b = b, (a % b)
    return a
```

5 Наименьшее общее кратное

Там, где возникает наибольший общий делитель, часто возникает и наименьшее общее кратное (НОК). Это наименьшее число, которое одновременно кратно и a, и b. Давайте подумаем, как его легко найти, используя имеющиеся знания. Для этого снова воспользуемся помощью математики.

Давайте посмотрим на разложения на простые множители чисел a и b:

$$a = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}, \quad \alpha_i \geq 0$$

$$b = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_k^{\beta_k}, \quad \beta_i \geq 0$$

Возьмем некоторый простой множитель p_i, пусть он входит в число a со степенью α_i, а в число b со степенью β_i. Тогда для наибольшего общего делителя верно, что в его состав этот множитель должен входить со степенью $\min(\alpha_i, \beta_i)$, это максимальная возможная степень, чтобы и a, и b делились на НОД:

$$\text{НОД}(a, b) = p_1^{\min(\alpha_1,\beta_1)} \cdot p_2^{\min(\alpha_2,\beta_2)} \cdot \ldots \cdot p_k^{\min(\alpha_k,\beta_k)}$$

Аналогично для наименьшего общего кратного: в НОК множитель p_i должен входить со степенью $\max(\alpha_i, \beta_i)$, чтобы это было наименьшее число, и при этом кратное и a, и b:

$$\text{НОК}(a, b) = p_1^{\max(\alpha_1,\beta_1)} \cdot p_2^{\max(\alpha_2,\beta_2)} \cdot \ldots \cdot p_k^{\max(\alpha_k,\beta_k)}$$

Теперь давайте посмотрим на произведение НОД и НОК, а точнее на степень простого множителя p_i. Эта степень равна $\min(\alpha_i, \beta_i) + \max(\alpha_i, \beta_i) = \alpha_i + \beta_i$, ведь при перемножении чисел степени складываются. Получается, что:

$$\text{НОД}(a, b) \cdot \text{НОК}(a, b) = p_1^{\alpha_1+\beta_1} \cdot p_2^{\alpha_2+\beta_2} \cdot ... \cdot p_k^{\alpha_k+\beta_k}$$

У произведения чисел a и b тоже степень множителя p_i равна $\alpha_i + \beta_i$, а значит:

$$\text{НОД}(a, b) \cdot \text{НОК}(a, b) = ab$$

Это довольно интересное и полезное свойство. Например, из него следует, что НОК чисел a и b равно произведению этих чисел, деленному на их НОД, который мы уже умеем находить с помощью алгоритма Евклида:

$$\text{НОК}(a, b) = \frac{ab}{\text{НОД}(a, b)}$$