Летняя школа по компьютерным наукам - 2016

Арифметика

Михаил Густокашин, 2016

1 Делимость и делители

Отвлечемся от технических вещей и перейдем к программе курса математики начальной школы.

Простым числом называется натуральное число, большее 1, которое делится нацело только на себя и 1 (имеет два натуральных делителя). Составными числами, соответственно, называются все остальные натуральные числа, большие единицы.

Простые числа имеют множество полезных применений, а также и сами по себе нередко являются сутью задачи. Наиболее известное промышленное применение простых чисел — в шифровании **RSA** с открытым ключом.

В первую очередь нам необходимо уметь проверять, является ли число N простым. Т.е. нам необходимо узнать, существуют ли такие натуральные числа x, y ($1 < x, y < N$), что $x \times y = N$. Кроме того, условимся, что $x \leq y$, тогда можно сказать, что x меньше либо равен квадратному корню из числа N (если это условие не выполнено, то произведение будет заведомо больше N).

Таким образом, можно написать следующую функцию, которая будет довольно эффективно проверять число на простоту:

```
bool isprime(int n)
{
  for (int i = 2; i * i <= j; ++i)
    if (n % i == 0)
return false;
  return true;
}
```

Пользуясь теми же соображениями, очень просто написать функцию, находящую все разложения числа на два множителя. Для этого достаточно убрать проверку на равенство N двум и в заменить `return 0` на обработку двух найденных делителей `i` и `n/i`.

Оба этих алгоритма имеют сложность $O(\sqrt{N})$.

2 НОД и НОК. Элементы теории остатков

Наибольшим общим делителем (НОД) двух натуральных чисел называется такое максимальное натуральное число, которое является делителем и первого и второго числа. Наименьшим общим кратным (НОК) двух натуральных чисел называется такое минимальное натуральное число, которое делится нацело на оба этих числа.

Аналогично вводятся понятия НОД и НОК многих чисел. На практике НОД и НОК многих чисел считаются с помощью последовательно попарного подсчета НОД и НОК (т.е. считается НОД уже обработанной части и очередного числа).

Задачи, в которых необходимо подсчитать НОД или НОК возникают довольно часто, особенно для НОД. Так, например, в подсчете количества точек с целыми координатами на отрезке используется НОД x и y координат отрезка.

В младшей школе изучается алгоритм Евклида, который позволяет найти НОД двух чисел. В нем используются следующие соотношения:

1) НОД$(a, 0) = a$

2) НОД(a, b) =НОД$(a\%b, b)$ =НОД$(a, b\%a)$

Запишем функцию подсчета НОД (GCD — Greatest Common Divisor):

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

Этот алгоритм очень прост, его сложность в худшем случае, равна $O(\log N)$, где N — большее из чисел.

НОК можно искать исходя из соотношения НОД$(a, b)\times$НОК$(a, b) = a \times b$.

Существует еще один способ поиска НОД: бинарный алгоритм Евклида. Этот способ основывается на соотношениях НОД$(2 \times a, 2 \times b) = 2\times$НОД$(a, b)$, НОД$(2 \times a, 2 \times b + 1)$ =НОД$(a, 2 \times b + 1)$, НОД$(2 \times a + 1, 2 \times b + 1)$ =НОД$(2 \times (a - b), 2 \times b + 1)$. Хотя при ипользовании этих соотношений время написания программы и собственно поиска НОД несколько возрастет (хотя по прежнему сложность алгоритма будет составлять $O(\log n)$), этот способ намного удобнее при поиске НОД длинных чисел. Действительно, в обычном алгоритме Евклида необходима операция взятия остатка от деления длинных чисел, а в бинарном — намного более простые операции длинного вычитания и деления на 2.

В олимпиадных задачах довольно часто требуется найти что-либо «по модулю» какого-либо числа, т.е. подсчитать остаток от деления результата на заданное число. Теория чисел достаточно подробно изучается в школьном курсе математики и мы не будем углубляться в нее, взяв лишь несколько практически важных результатов.

В частности, нам важны следующие свойства остатков: $(a+b)\%n = (a\%n+b\%n)\%n$ и $(a \times b)\%n = (a\%n \times b\%n)\%n$. Эти свойства часто бывают полезными, т.к. обычно в задачах, где требуется подсчитать остатки, возникают довольно большие числа, и уменьшение размерности операндов в промежуточных вычислениях намного облегчает задачу.

Также довольно часто требуется найти удовлетворяющий условию элемент какой-либо последовательности по модулю данного числа. В этом случае можно составить «таблицу умножения» по модулю n или другую таблицу с правилами перехода — это может значительно облегчить решение задачи.

3 Разложение числа на простые множители

Вернемся к простым числам. Любое число можно представить в виде произведения простых чисел, причем это представление будет единственно.

Обычно, такое представление выглядит в виде одномерного массива, содержащего простые числа, и еще одного одномерного массива для непосредственного представления числа, в каждой ячейке которого содержится степень соответственного просто числа.

Например, пусть у нас есть массив простых чисел $[2, 3, 5, 7, 11, 13]$, тогда массив с представлением числа 2600 будет выглядеть как $[3, 0, 2, 0, 0, 1]$ из которого можно получить $2^3 \times 5^2 \times 13^1 = 2600$.

Такое представление неоправданно генерировать для одного числа (если это не является необходимым условием для решения задачи). Однако, если чисел много, то приведение их в такое представление и обратно может быть весьма полезным.

Например, с помощью такого представления очень просто реализовать умножение. Рассмотрим наше число 2600 и число 11858, которое представляется массивом $[1, 0, 0, 2, 2, 0]$. Чтобы получить произведение этих чисел, достаточно сложить соответствующие элементы массивов. Результатом будет массив $[4, 0, 2, 2, 2, 1]$, т.е. $2^4 \times 5^2 \times 7^2 \times 11^2 \times 13^1 = 30830800$, что совпадает с результатом умножения.

После некоторых размышлений можно также реализовать деление с остатком, но в рамках лекции мы этого делать не будем, желающие могут сами придумать алгоритм.

Из этого представления также можно получить НОД и НОК этих чисел.

Для нахождения НОД надо выбирать минимум из степеней (это логично, т.к. число X^n кратно X^k, если $n \geq k$). Для подсчета НОК надо брать максимум из степеней. Этот же метод работает для подсчета НОД и НОК произвольного количества чисел.

Для наших чисел 2600 и 11858, НОД, подсчитанный таким образом, представим массивом $[1, 0, 0, 0, 0, 0]$, т.е. равен 2, а НОК — массивом $[3, 0, 2, 2, 2, 1]$ и равен 15415400. С помощью алгоритма Евклида несложно убедиться, что результат верен.

Кроме того, у разложения числа на простые множители существуют более специфичные назначения, которые встретятся по ходу решения практических туров.

Нахождение всех простых чисел до N занимает $O(N \times \sqrt{N})$ времени, подсчет степеней для одного числа занимает около $O(N)$ времени (если степени не слишком большие), и операции умножения, нахождения НОД и НОК занимают также $O(N)$ времени. Таким образом, суммарное время составляет около $O(N \times \sqrt{N} + N \times M)$, где N — максимальное из чисел, а M — количество этих чисел.

В некоторых случаях нам необходимо разложить только одно число на простые множители. В такой ситуации наиболее эффективным методом будет модификация функции проверки числа на простоту. Действительно, если каждый раз при нахождении делителя мы будем делить разлагаемое число на него до тех пор, пока оно делится (и запоминать степень, с которой этот делитель входит в разложение числа), то мы получим все простые делители кроме, возможно, одного. Действительно, поиск делителей необходимо осуществлять до квадратного корня из числа, а в случае, если после деления осталась не единица — этот остаток также является простым делителем, причем степень его вхождения всегда равна 1. Например, 26 представляется как $2^1 \times 13^1$, где 13 — единственный делитель, больший квадратного корня.

4 Быстрое возведение в степень

Довольно часто возникает задача быстрого возведения числа или другого объекта, для которого определена операция умножения, в какую-либо степень. Наивный алгоритм, когда мы просто нужное число раз умножаем число на само себя, имеет сложность $O(N)$, где N — показатель степени.

При возведении в степень число растет очень быстро (а значит, наверняка требуются операции с длинными числами). Поэтому научиться возводить число в степень быстрее, чем за $O(N)$ — достаточно важная задача.

Рассматривая степени некоторых чисел, можно догадаться о методе, которым следует пользоваться. Например, для возведения 3 в 4-ю степень нам нужно проделать 3 операции умножения. Однако, если изменить порядок действий на такой: $(3^2)^2$, то потребуется всего два умножения. Следует помнить, что при возведении числа в степени еще в какую-либо степень показатели степеней перемножаются. Именно на сокращении четных степеней основывается идея быстрого возведения в степень.

Приведем текст функции, которая возводит число a в степень n:

```
int pow(int a, int n)
{
if (n == 0)
return 1;
if (n % 2 == 0)
return pow(a * a, n / 2);
return a * pow(a * a, (n - 1) / 2);
}
```

В англоязычной литературе алгоритм быстрого возведения в степень обзывают забавным словосочетанием “Russian peasant algorithm”, что переводится как «алгоритм русского крестьянина».