ВАРИАНТ 37091 - Решение

1. В теории чисел натуральное число называется B-гладким, если все его простые делители не превосходят B. Разработайте алгоритм проверки чисел в диапазоне от P до Q на B-гладкость.

Решение. Построим массив простых чисел с помощью решета Эратосфена. Далее для каждого числа *n* из диапазона от P до Q проверяем, являются ли простые числа, большие B, но меньшие или равные *n* / 2, делителями числа *n*. Если это так, то число не является B-гладким. В противном случае число является B-гладким.

Для использования решета Эратосфена необходимо построить массив чисел в заданном диапазоне от 1 до Q / 2. Поскольку число 1 не является простым, в элемент массива с индексом 1 занесём значение 0 – будет удобнее, если индекс массива и число в массиве совпадают. Затем для чисел *i* в диапазоне от 2 до $\sqrt{Q/2}$, начиная с числа *i*, вычёркиваем из массива (заменяем нулями) все числа с шагом *i* (само число *i* не вычёркивается). Для нахождения следующего значения *i* нужно найти первый незачёркнутый (ненулевой) элемент массива после текущего значения *i*.

```
алг B-гладкость()
нач
  цел p, q, b, primes[q / 2], n, i, j
  лог is_smooth

  ввод p, q, b

  если p < 1 или q < 1 или b < 1 или p > q то
    вывод "Некорректные исходные данные"
  иначе

    // Построение массива простых чисел
    primes[1] = 0
    для i от 2 до q / 2
    нц
      primes[i] = i
    кц

    i = 2
    пока i <= целая_часть(sqrt(q / 2))
    нц
      для j от 2 * i до q / 2 шаг i
      нц
        primes[j] = 0
      кц
      выполнить
        i = i + 1
      до primes[i] <> 0
    кц

   // Проверка чисел от p до q на b-гладкость
   для n от p до q
   нц
     is_smooth = истина
     i = b + 1
     пока i <= n / 2 и is_smooth
     нц
       если primes[i] <> 0 и n mod i = 0 то
         is_smooth = ложь
       всё
       i = i + 1
     кц
     если is_smooth то
       вывод n, " является ", b, "-гладким числом"
     иначе
       вывод n, " не является ", b, "-гладким числом"
     всё
   кц

  всё
кон
```

2. Число n представляется в виде произведение двух простых чисел $n = p \cdot q$. Составьте алгоритм для нахождения этих чисел, если известно, что $n = 40003200063$, а $|p - q| = 2$.

Решение.

1 способ.

1. Из условия следует, что $(q + 2) \cdot q = n$ или $(q - 2) \cdot q = n$. Имеем 2 квадратных уравнения $q^2 + 2q - n = 0$ и $q^2 - 2q - n = 0$.
2. Вычисляем значение дискриминанта для этих уравнений: $4 + 4n$.
3. Вычисляем решения этих уравнений $x_1 = -1 + \sqrt{n+1}$, $x_2 = 1 + \sqrt{n+1}$, $x_3 = -1 - \sqrt{n+1}$, $x_4 = 1 - \sqrt{n+1}$.
4. Если $n \geq 0$ и x_1 и x_2 – целые числа, то $q = x_1$ и $p = x_2$ (или наоборот).
5. Проверим найденные числа на простоту. Может оказаться, что одно или оба не являются простыми, и задача не имеет решения.

2 способ.

1. Поскольку $|p - q| = 2$, то есть некоторое x такое, что $p = x - 1$, а $q = x + 1$ (или наоборот).
2. Тогда $n = (x - 1) \cdot (x + 1)$, $n = x^2 - 1$.
3. Следовательно, $x = \sqrt{n+1}$, и $p = \sqrt{n+1} - 1$, а $q = \sqrt{n+1} + 1$ (или наоборот).
4. Проверим найденные числа на простоту. Может оказаться, что одно или оба не являются простыми, и задача не имеет решения.

Чтобы проверить, что число n является простым, необходимо перебрать его возможные делители от 2 до $\sqrt{n}$ и проверить, делится ли число n на какой-либо из возможных делителей. Если это так, то число не является простым. При этом число 2 является простым, а число 1 не является простым.

Для сокращения перебора можно сначала проверить делимость числа n на 2, а потом проверить его делимость на возможные нечётные делители от 3 до $\sqrt{n}$ с шагом 2. Не забываем, что само число 2 является простым.

Можно ещё больше сократить перебор, отбрасывая числа, которые делятся на 2 и на 3. Для этого для числа n проверяются возможные делители i от 5 до $\sqrt{n}$ с шагом 6, но на каждом шаге цикла проверяется делимость числа n на i и на $(i + 2)$, т.е. получается ряд 5, 7, 11, 13 и т.д. До цикла надо проверить делимость числа n на 2 и на 3, а также надо учесть, что сами числа 2 и 3 являются простыми.

3. В теории чисел задача Знама спрашивает, какие множества k целых чисел имеют свойство, что каждое целое в множестве является собственным делителем произведения других целых чисел в множестве плюс 1. То есть, если дано число k, какие существуют множества целых чисел $\{n_1, ..., n_k\}$ таких, что для любого i число n_i делит, но не равно $\left(\prod_{j \neq i}^{k} n_j + 1\right)$.

Разработайте алгоритм нахождения числа решений задачи Знама для k в диапазоне от P до Q. Принять верхнюю границу $n_i = 1000000$.

Решение. Задача решается перебором всех вариантов. Но надо сформировать все наборы значений из k чисел. Поскольку k заранее неизвестно, невозможно написать несколько циклов. Будем использовать следующий алгоритм. Сначала формируем массив из k элементов, каждый из

которых равен минимальному возможному значению (в данном случае это 2). Далее увеличиваем последний элемент массива на 1, пока он не станет равным максимальному значению. После этого последнему элементу снова присваиваем минимальное значение, а предыдущий элемент увеличиваем на 1. Если предыдущий элемент (или несколько предыдущих) равен (равны) максимальному значению, присваиваем ему (им) минимальное значение, а на 1 увеличиваем первый с конца элемент, который не равен максимальному значению. Когда все элементы массива будут равны максимальному значению, заканчиваем перебор.

В наборе чисел, удовлетворяющем условию задачи Знама, может быть только одно чётное число, т.к. иначе значение выражения $\left(\prod_{j\neq i}^{k} n_j + 1\right)$ будет нечётным, и если n_i – чётное число, то оно точно не будет делителем этого выражения. Поэтому каждый сформированный набор можно проверить на наличие чётных чисел и все чётные элементы, кроме первого, увеличить на 1 и установить шаг изменения последнего элемента равным 2. Если первые (k – 1) элементы – нечётные, оставляем шаг изменения последнего элемента равным 1.

Также в наборе не может быть одинаковых чисел, т.к. в этом случае выражение $\left(\prod_{j\neq i}^{k} n_j + 1\right)$ также не будет делиться на один из сомножителей произведения.

Следовательно, первая подходящая комбинация состоит из чисел 2, 3, 5, 7 и т.д. Кроме того, при переходе к следующему возможному набору надо учитывать, что каждое следующее число должно быть больше предыдущего, т.к. это позволит нам не рассматривать наборы, состоящие из одинаковых чисел, расположенных в разном порядке.

```
алг ЗадачаЗнама()
нач
  цел p, q, k, r, step, i, j
  цел n[q]
  лог all, found
  цел константа limit = 1e11

  ввод p, q

  для k от p до q
  нц
    r = 0

    // Заполняем массив исходными значениями. Первый элемент массива положим равным 2, а остальные – 3, 5 и т.д.
    n[1] = 2
    для i от 2 до k
    нц
      n[i] = 2 * i – 1
    кц

    step = 2
    all = ложь
    пока не all
    нц
      если УдовлетворяетУсловию(n, k) то
        r = r + 1
      всё

      // Переход к следующему набору
      если n[k] + step <= limit то          // если можно изменить последний элемент набора
        n[k] = n[k] + step                  // меняем его
      иначе                                 // ищем первый с конца элемент, который можно изменить
        found = ложь
        i = k – 1
        пока i >= 1 и не found
        нц
          если n[i] + 1 <= limit – ((k – i) * 2 – 1) то
           found = истина
          иначе
            i = i – 1
          всё
        кц
        если не found то                    // если нет элемента, который можно изменить
          all = истина                      // завершаем цикл по перебору комбинаций
```

```
        иначе
          n[i] = n[i] + 1
          found = ложь                      // проверяем наличие чётного элемента до i-го элемента
          j = i - 1
          пока j >= 1 и не found
          нц
            если n[j] mod 2 = 0 то
              found = истина
            всё
            j = j - 1
          кц
          если found то                     // если до i-го элемента есть чётный элемент
            если n[i] mod 2 = 0 то          // если i-ый элемент тоже чётный
              n[i] = n[i] + 1               // делаем его нечётным
            всё
            для j от i + 1 до k             // остальным элементам присваиваем следующие нечётные значения
            нц
              n[j] = n[j - 1] + 2
            кц
            step = 2                        // шаг = 2, т.к. есть чётный элемент, остальные должны быть нечётным
          иначе                             // если до i-го элемента нет чётного элемента
            если n[i] mod 2 = 0 то          // если i-ый элемент является чётным
              n[i + 1] = n[i] + 1           // (i + 1)-ому элементу присваиваем следующее (нечётное) значение
              для j от i + 2 до k           // остальным элементам присваиваем следующие нечётные значения
              нц
                n[j] = n[j - 1] + 2
              кц
              step = 2                      // шаг = 2, т.к. есть чётный элемент, остальные должны быть нечётным
            иначе                           // если i-ый элемент является нечётным
              n[i + 1] = n[i] + 1           // присваиваем следующему элементу следующее (чётное) значение
              если i + 2 <= k то            // ещё следующему (если он есть) присваиваем нечётное значение
                n[i + 2] = n[i + 1] + 1
              всё
              для j от i + 3 до k           // остальным элементам  присваиваем следующие нечётные значения
              нц
                n[j] = n[j - 1] + 2
              кц
              если n[k] mod 2 = 0 то        // если чётным оказался последний элемент, значит, все предыдущие
                step = 1                    // являются нечётными, и шаг = 1
              иначе
                step = 2
              всё
            всё
          всё
        всё
      всё
    кц

    вывод "Количество решений для набора из ", k, " чисел равно ", r
  кц
кон

алг УдовлетворяетУсловию(арг цел n[k], арг цел k)
нач
  цел i, j, p
  лог f

  f = истина
  i = 1
  пока i <= k и f
  нц
    p = 1
    для j от 1 до i - 1
    нц
      p = p * n[j]
    кц
    для j от i + 1 до k
    нц
      p = p * n[j]
    кц
    если (p + 1) = n[i] или (p + 1) mod n[i] <> 0 то
      f = ложь
    всё
    i = i + 1
  кц
  вернуть f
кон
```

4. В теории чисел уникальное простое число – это определённый вид простых чисел. Простое число $p \neq \{2, 5\}$ называется уникальным, если не существует другого простого q, такого что длина периода разложения в десятичную дробь обратной величины, $1 / p$, равна длине периода $1 / q$. Разработайте алгоритм, определяющий для простых чисел в диапазоне от U до W, являются ли они уникальными простыми. Ограничить сверху $q = 10^9$.

Решение. Определим период дроби $1 / p$ (если он есть). Для этого будем делить 10 на p, брать остаток от деления, умножать его на 10 и снова делить на p. Когда остаток от деления станет равен 1, то в дроби начнётся новый повтор периодических цифр. Надо подсчитать, сколько операций деления потребовалось для получения остатка, равного 1. Это и будет период дроби. Если дробь не является периодической, процесс деления будем останавливать, когда количество делений станет в 100 раз больше, чем количество цифр в числе p, т.к. период дроби не должен принципиально отличаться от количества цифр в знаменатели дроби. Для определения количества цифр в числе можно последовательно сравнивать его с числами 10, 100, 1000 и т.д.

Построим массив простых чисел от 1 до 10^9 (или до W, если $W > 10^9$). Вычеркнем из полученного массива числа 2 и 5. Для всех полученных простых чисел найдём период соответствующей дроби (если дробь не периодическая, можно записать 0). Затем для чисел p в диапазоне от U до W найдём число q, период дроби которого равен периоду дроби $1 / p$. Если такого числа нет, то число p является уникальным простым числом.

Построить массив простых чисел можно с помощью решета Эратосфена. Но этот алгоритм предполагает начальное заполнение массива всеми числами, т.е. простые числа будут размещаться в массиве не компактно и останется много неиспользуемых элементов. Для того чтобы уменьшить размер массива, можно использовать другой алгоритм нахождения простых чисел.

В первый элемент массива запишем число 2 (это первое простое число). Далее для всех нечётных чисел n от 3 будем проверять, делится ли число n на уже найденные простые числа до $\sqrt{n}$. Если число n ни на что не делится, значит, оно является простым, и нужно дописать его в массив. Поиск простых чисел заканчивается при заполнении массива. Хотя этот алгоритм работает дольше, чем решето Эратосфена, он позволяет компактно записать простые числа в массив и сократить размер массива (или увеличить количество рассматриваемых простых чисел).

```
алг УникальныеПростыеЧисла()
нач
  цел константа nmax = цел(1e9)
  цел u, w, primes[nmax], periods[nmax], n, i, j, steps, num, length
  лог found

  ввод u, w

  если u < 1 или w < 1 или u > w то
    вывод "Некорректные исходные данные"
  иначе

    // Построение массива простых чисел
    primes[1] = 2
    n = 1
    num = 3
    пока n < nmax
    нц
      found = ложь
      i = 1
      пока i <= n и primes[i] <= целая_часть(sqrt(num)) и не found
      нц
        если num mod primes[i] = 0 то
          found = истина
        всё
        i = i + 1
      кц
```

```
  если не found то
    n = n + 1
    primes[n] = num
  всё
  num = num + 2
кц

// Обнуляем период дроби для чисел 2 и 5
periods[1] = 0
periods[3] = 0

// Считаем период дроби для числа 3
steps  = 0
num    = 10
found  = ложь
length = Длина(primes[2])
пока не found и steps < length * 100
нц
  steps = steps + 1
  num = num mod primes[2]
  если num = 1 то
    found = истина
  иначе
    num = num * 10
  всё
кц
если found то
  periods[2] = steps
иначе
  periods[2] = 0
всё

// Считаем период дроби для остальных чисел
для i от 4 до n
нц
  steps  = 0
  num    = 10
  found  = ложь
  length = Длина(primes[i])
  пока не found и steps < length * 100
  нц
    steps = steps + 1
    num = num mod primes[i]
    если num = 1 то
      found = истина
    иначе
      num = num * 10
    всё
  кц
  если found то
    periods[i] = steps
  иначе
    periods[i] = 0
  всё
кц

// Ищем дроби с одинаковым периодом
i = 1
пока primes[i] < u
нц
  i = i + 1
кц
пока primes[i] <= w
нц
  found = ложь
  если periods[i] <> 0 то
    j = 1
    пока j <= i – 1 и не found
    нц
      если periods[i] = periods[j] то
        found = истина
      всё
      j = j + 1
    кц
    j = i + 1
    пока j <= n и не found
    нц
      если periods[i] = periods[j] то
        found = истина
      всё
      j = j + 1
    кц
  всё
```

```
      если не found то
        вывод primes[i], " является уникальным простым числом"
      всё
      i = i + 1
    кц
  всё
кон

алг Длина(арг цел n)
нач
цел limit, len

  limit = 10
  len = 1
  пока истина
  нц
    если n < limit то
      вернуть len
    всё
    limit = limit * 10
    len = len + 1
  кц
кон
```

Возможен другой подход к решению этой задачи. Период дроби 1 / *p* равен 1, если остаток от деления 10 на *p* равен 1. В этом случае при следующем делении мы получаем такой же результат. Период дроби 1 / *p* равен 2, если остаток от деления 100 на *p* равен 1, и т.д. Таким образом, чтобы период дроби был равен 1, $p \cdot x$ (*x* – результат деления 10 на *p*) должно быть равно 9. Чтобы период дроби был равен 2, $p \cdot x$ должно быть равно 99, и т.д. Для периода дроби, равного 1, мы получаем единственное решение $p = x = 3$. Рассмотрим ряд 99, 999, 9999… Все эти числа делятся на 9. Само число 9 не является простым, а для числа 3 период дроби равен 1. Следовательно, числа, для которых период дроби равен 2, 3, 4…, должны быть простыми делителями чисел 11, 111, 1111… Поэтому получаем следующий алгоритм.

Построим массив простых чисел. Найдём периоды дробей для этих чисел. Далее, будем искать уникальное простое число с периодом дроби 2, 3, 4… Для этого рассматривая ряд 11, 111, 1111…, будем искать простые делители для каждого числа этого ряда, и если среди простых делителей некоторого числа из ряда будет только один с соответствующим периодом дроби, значит, этот делитель и является уникальным простым числом с соответствующим периодом дроби. Если таких делителей несколько, то нет уникального простого числа с таким периодом дроби.

Можно обойтись без построения массива простых чисел, что на самом деле является долгим процессом. Рассмотрим числа из ряда 11, 111, 1111… и для каждого числа будем последовательно искать простые делители. Для каждого найденного простого делителя будем искать период дроби и считать, сколько делителей имеют нужный период дроби (2, 3, 4…). Если такой делитель только один, значит, он является уникальным простым числом с заданным периодом дроби.

```
алг УникальныеПростыеЧисла()
нач
  цел pi, pl, num, num2, f, res, period, k

  ввод pi, pl

  если pi = 1 то
    вывод "3 является уникальным простым числом с периодом 1"
    pi = pi + 1
  всё

  num = 1
  для k от 2 до pi
  нц
    num = num * 10 + 1
  кц

  для period от pi до pl
  нц
    num2 = num                                  // Сохраняем число из нужного количества единиц
```

```
    k = 0                                                 // Количество делителей с нужным периодом
    f = 3                                                 // Первый делитель равен 3
    пока num2 > 1 и k <= 1
    нц
      если num2 mod f = 0 то                              // Если число делится на текущий делитель
        если Период(f) = period то                        // Если период дроби делителя равен нужному значению
          k = k + 1
          res = f
        всё
        пока num2 mod f = 0                               // Несколько раз делим число на текущий делитель
        нц
          num2 = num2 div f
        кц
      иначе                                               // Если число не делится на текущий делитель,
        если f > целая_часть(sqrt(num2)) то               // но текущий делитель больше корня из числа,
          если Период(num2) = period то                   // значит, число является простым
            k = k + 1                                     // Считаем период дроби для этого числа
            res = num2
          всё
          num2 = 1                                        // Делим число на само себя
        всё
      всё
      f = f + 2
    кц
    если k = 1 то
      вывод res, " является уникальным простым числом с периодом ", period
    иначе
      вывод "Нет уникального простого числа с периодом ", period
    всё
    num = num * 10 + 1
  кц
кон

алг Период(арг цел prime)
нц
  цел steps, num, length
  лог found

  steps  = 0
  num    = 10
  found  = ложь
  length = Длина(prime)
  пока не found и steps < length * 100
  нц
    steps = steps + 1
    num = num mod prime
    если num = 1 то
      found = истина
    иначе
      num = num * 10
    всё
  кц
  если found то
    вернуть steps
  иначе
    вернуть 0
  всё
кон

алг Длина(арг цел n)
нач
цел limit, len

  limit = 10
  len = 1
  пока истина
  нц
    если n < limit то
      вернуть len
    всё
    limit = limit * 10
    len = len + 1
  кц
кон
```

5. Широко известна древнегреческая задача об Ахилле и черепахе.

Предположим, что Ахилл бежит вдесятеро быстрее, чем ползёт черепаха, и в начале состязания черепаха имеет 100 м форы. К тому времени, когда Ахиллес пробежит 100 м, черепаха успеет проползти 10 м. Когда же Ахилл пробежит и эти 10 м, черепаха уползёт вперёд на 1 м, и так далее. Таким образом, черепаха всегда будет впереди Ахилла и он её никогда не догонит.

Вы делаете расчёт данной задачи на ЭВМ. Какой будет получен результат в ответе?

Решение (один из вариантов). Поскольку количество знаков в представлении чисел на ЭВМ ограничено, существуют такое понятие, как *машинный ноль*. Порог машинного нуля – числовое значение с таким отрицательным порядком, которое воспринимается машиной как ноль.

Если мы будем рассматривать путь, пройденный Ахиллом и черепахой за некоторый период времени, то мы увидим, что этот путь всё время уменьшается, и в какой-то момент времени путь, пройденный черепахой, станет равным машинному нулю, а путь, пройденный Ахиллом, ещё не станет равным машинному нулю. Как результат, положение Ахилла и черепахи в пространстве для ЭВМ совпадут. В этот момент Ахилл догонит черепаху.