Занятие №3. Решение задач из области арифметики

К основному тексту каждого занятия приложениями следуют «Задачи» с текстами всех задач, предлагаемых на практической части, «Подсказки» с ответами на вопросы в тексте занятия, подробные «Разборы» всех предлагаемых задач и «Справочник» с листингами всех важных алгоритмов занятия.

«Арифметика» - греческое слово, которое в Толковом словаре живого великорусского языка Владимира Даля означает «учение о счете, наука о счислении; основа всей математики…» В современном издании вместо многоточия должно стоять «и программирования»! Трудно придумать задачу по программированию, которая никак не касается арифметики. Да и возможно ли? Ведь компьютер фактически умеет только считать, хотя делает это очень быстро.

На этом занятии мы поговорим о свойствах чисел, которые часто используются в программировании.

Прежде всего, это «делимость». (Это настолько важное понятие, что мы посвятим делителям целое занятие.)

Проверку того, что одно число делится на другое, обычно делается в программировании нахождением остатка. В языке Java для этого есть специальная операция %. Чему, например, равно 5 % 2? А 9 % 10? Если остаток не равен нулю - не делится, если равен – делится.

Проверка на четность

```
if (x % 2 == 0)
{
     // число четное!
}
```

Заметим, что *для определения нечетности лучше сравнивать именно с нулем, а не с единицей*, потому что для нечетных отрицательных чисел остаток равен… Напишите мини-программу и проверьте чему!

Часто бывает нужно проводить операции с остатками. Говорят, что **числа сравнимы по модулю m, если они дают одинаковый остаток от деления на m**. Или, что тоже – их разность делится на m. Записывают это при помощи специального знака равенства с

тремя палочками $a \equiv b \pmod m$. Например, $17 \equiv 11 \pmod 3$, потому что и 17 и 11 дают остаток 2 при делении на 3.

Приведем некоторые полезные свойства «арифметики остатков» без доказательств. Попробуйте в свободное от программирования время их доказать. В Подсказке 3.1 рассматривается значительно больше свойств и там они приведены с доказательствами.

Немного теории

Арифметика остатков

Свойство транзитивности. Если $a \equiv b \pmod m$, $b \equiv c \pmod m \Rightarrow a \equiv c \pmod m$.

Отсюда следует, что в свойствах сравнений выражение $a \equiv b \pmod m$ можно читать и как «a дает остаток b при делении на m».

Свойство сложения. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то $a+c \equiv b+d \pmod m$.

В частности если $a \equiv b \pmod m$,то $a+n \equiv b+n \pmod m$.

Свойство вычитания. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то $a-c \equiv b-d \pmod m$.

Свойство произведения. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то $ac \equiv bd \pmod m$.

В частности если $a \equiv b \pmod m$,то $an \equiv bn \pmod m$.

Свойство степени. Если $a \equiv b \pmod m$, то $a^n \equiv b^n \pmod m$.

Например, часто в задачах по программированию требуется найти не полный ответ, а именно остаток от деления. Приведенные свойства как раз и позволяют сделать это.

Например, нам нужно узнать остаток от деления числа **`a`** возведенного в степень **`b`** на данное число **`c`** (все числа типа **`int`**). Проблема в том, что эта степень получается очень большой – «не влезает» в стандартный целочисленный тип. Конечно, некоторые могут попробовать использовать **`BigInteger`** – просто посчитать, а потом взять остаток от деления функцией **`mod`**, но это совсем не эффективно – работа с большими числами требует большого количества действий и может привести к ошибке Time Limit. А полученная степень может содержать огромное число знаков, что в результате приведет к Memory Limit. Но благодаря свойствам остатков эту задачу можно решить эффективно и ограничится использованием типа **`int`**. Степень можно реализовать последовательными умножениями. Будем вычислять степень p последовательными умножениями на a. По последнему свойству если $p \equiv k \pmod c$, что значит p при делении на c дает остаток k, тогда $pa \equiv ka \pmod c$. То есть, если на очередном этапе

Два пастуха продали свое стадо, выручив за каждую овцу столько тугриков, сколько овец было в стаде. Все деньги они обменяли на купюры по 10 тугриков и несколько монет.

Пастухи хотели поделить выручку поровну, но одному досталось на купюру больше. Тогда, чтобы было справедливо, он отдал товарищу свой перочинный ножик.

Сколько стоил ножик?

См. Подсказку 3.7

p становится больше c, просто сразу можно заменить его остатком (числом k) от деления на c. Если учесть, что остаток от деления меньшего числа на большее равен меньшему числу, то остаток можно постоянно брать без проверок. Код в таком случае получается очень простым:

```
int p = 1;
for (int i = 0; i < b; i++)
{
     p = p * a;
     p %= c; // взять остаток и положить его сразу в p
}
```

Аналогичная идея используется в решении интересной задачи этого занятия «Полярные единички», (прочитайте ее условие на стр.9). Если мы знаем остаток от деления числа из N единичек на данное в задаче число, то как узнать остаток от деления числа, которое записывается N+1 единичкой? (Посмотрите Подсказку 3.2) А как узнать, что остаток 0 никогда не получится?

Цифры числа

Для человека естественно, что числа состоят из цифр. Любой скажет, что третья цифра числа 257038 – это 7. Для компьютера это непростая задача. Число в памяти хранится в двоичном виде и для ответа нужно совершить много операций. Попробуем выяснить каких.

Проще всего «вычислить» последнюю цифру. Как? (Подсказка 3.3)

Чтобы «вычислить» вторую с конца цифру, можно свести задачу к предыдущей. «Отбросим» последнюю цифру и снова ее найдем последнюю. Отбросить можно так: вычтем последнюю цифру из числа и поделим его на 10. Но ведь деление целых чисел в Java происходит нацело, например 275/10 это 27 ровно! Значит, даже не надо вычитать, а просто поделить нацело. Будем так делать в цикле, получая цифры числа **x** справа-налево:

Получение цифр числа

```
while(____)
{
     c = x % 10;
     // В переменной c очередная цифра –
     // делаем что-нибудь с ней...
     x /= 10; // уменьшаем переменную x в 10 раз
}
```

Каково условие цикла? (Подсказка 3.4)

(В процессе работы с курсом предполагается активная работа слушателей с пособием. В листингах, приводимых в тексте занятий, часто присутствуют пропуски-подчеркивания, которые надо заполнить карандашом самостоятельно. Правильные варианты при этом можно найти в Подсказках или Справочнике)

Нужно только отдельно рассматривать случай, когда в переменной x изначально лежит ноль.

Заметим, что если делить не на 10, а на другое число, скажем, p, то мы получим цифры нашего числа, записанного в p-ичной системе счисления. Подробнее мы поговорим об этом в одном из следующих модулей курса.

Очень большую роль в арифметике и программировании играют простые числа. Достаточно сказать, что «основная теорема арифметики» именно о простых числах. Она утверждает, что любое натуральное число раскладывается в произведение простых чисел единственным образом с точностью до порядка сомножителей. Если же расположить числа в определенном порядке, например, по возрастанию, единственность становится абсолютной. Например, 1176 = 2*2*2*3*7*7 = 2^3*3*7^2. Именно в таком виде вам нужно представить заданное число в задаче «Разложение на простые++».

Но сначала о проверке на простоту.

Чтобы проверить число N на простоту можно просто перебрать все числа, меньшие N:

```
boolean isPrime = true;
for (int d = __; d < N; d++)
{
     if (N % d == 0)
     {
          isPrime = false;
     }
}
```

Но, как мы уже говорили на первом занятии это очень долго. Нужно ли проверять делится ли число 101 на 60? Конечно, нет. Но и на 50 (почти на половину) проверять нет необходимости. Если бы оно делилось, то мы бы узнали об этом значительно раньше. Когда? Еще при проверке деления на два! Таким образом, достаточно перебрать только «меньшие» возможные делители, то есть такие d, что если d*k = N, то d ≤ k. Это очень существенно сократит перебор. Для числа 101 мы должны будем проверять только до 11. Для 1000001 – до 1001. Знакомые с понятием арифметического корня скажут, что **надо проверять до корня из N включительно**. Оформим решение в виде функции:

Проверка на простоту

```
boolean isPrime(int N)
{
     int d = 2; // начинаем проверять с двух
```

```
    boolean result = true; // «презумпция простоты»
    while (d <= N / d) // парный делитель к d равен N / d
    {
        if (N % d == 0) // d – делитель
        {// ***
            result = false; // составное!
            break; // немедленно выходим из цикла
        }// ***
        d++;
    }
    return result_____________;
}
```

Правильно ли работает эта функция для числа два (это единственное четное простое число)? Почему? А для единицы? (См. Подсказку 3.5). Исправьте, допишите код, если это необходимо.

Подобный цикл работает и для решения задач на нахождение суммы, произведения, количества делителей. Что надо вписать в тело if для этого?

Сумма делителей

```
if (N % d == 0) // d – делитель
    {// ***
        ___________________
        ___________________
        ___________________
    }// ***
```

Количество делителей

```
if (N % d == 0) // d – делитель
    {// ***
        ___________________
        ___________________
        ___________________
    }// ***
```

В задаче «Почти простые числа» нужно определить, раскладывается ли число в произведение двух неравных простых. Можно попробовать разложить число на два множителя, проверяя каждый из них на простоту. Но это не проходит по времени. Во-первых, данное число может быть очень большим, во-вторых, проверка на простоту занимает немалое время, и получается два цикла в цикле. А можно просто посчитать количество нетривиальных делителей (все без учета единицы и самого числа) и мгновенно получить ответ! Как, см. Подсказку 3.6.

Как разложить число на простые множители?

Можно воспользоваться следующим трюком: если уменьшать число, деля его на очередной найденный делитель до тех пор, пока можно на него делить – то среди найденных делителей будут только простые.

Например, возьмем число 100:

100 делится на 2 -> найден простой делитель 2 – выводим,

делим, получаем 50, делим, выводим, получаем 25 – на 2 не делится – увеличиваем d

3 – не делится – увеличиваем d

4 – не делится (две двойки «вышли» раньше)

5 – делится -> простой делитель 5 – выводим его,

делим, получаем 5, делим, получаем 1

конец алгоритма

В коде это будет выглядеть так:

Разложение на простые множители

```
int d = 2; // начинаем проверять с двух
while ( N > 1) // пока не уменьшим до 1 – больше простых
делителей нет
{
     if (N % d == 0) // d – делитель
     {
          _______________
          _______________
     }
     d++;
}
```

Заметим, что этот цикл работает долго для простых чисел. Он перебирает все числа до самого числа. Для ускорения нужно воспользоваться тем же трюком, выходить из цикла, когда перебираемый кандидат-делитель станет большим.

Задачи

Задача A. Уравнение

Решить в целых числах уравнение *ax+b=0*.

Вводятся 2 числа: *a* и *b*. Необходимо вывести все решения, если их число конечно, “NO” (без кавычек), если решений нет, и “INF” (без кавычек), если решений бесконечно много.

Примеры

Входные данные	Выходные данные
1 1	-1
2 1	NO

Задача B. Фишки

В каждую крайнюю клетку квадратной доски поставили по фишке. Могло ли оказаться, что выставлено ровно *k* фишек? (Например, если доска 2 x 2, то выставлено 4 фишки, а если 6 x 6 – то 20).

Вводится одно натуральное число k.

Примеры

Входные данные	Выходные данные
20	YES
13	NO

Задача C. Мороженое

В кафе мороженое продают по три шарика и по пять шариков. Можно ли купить ровно *k* шариков мороженого?

Примеры

Входные данные	Выходные данные
3	YES
1	NO

Задача D. Автобусы

Источник: Московская олимпиада, 7-9 класс 2008 года

Для заезда в оздоровительный лагерь организаторы решили заказать автобусы. Известно, что в лагерь собираются поехать *N* детей и *M* взрослых. Каждый автобус вмещает *K* человек. В каждом автобусе, в котором поедут дети, должно быть не менее двух взрослых.

Определите, удастся ли отправить в лагерь всех детей и взрослых, и если да, то какое минимальное количество автобусов требуется для этого заказать.

Формат входных данных

На вход программы поступают 3 натуральных числа, записанных через пробел - *N, M* и *K*, каждое из них не превосходит 10 000.

Формат выходных данных

Выведите количество автобусов, которые нужно заказать. Если же отправить всех в лагерь невозможно, выведите 0 (ноль).

Примеры

Входные данные	Выходные данные
10 4 7	2
10 4 5	0

Задача E. Количество нулей

Дано натуральное число N. Напишите функцию `int NumberOfZeroes(int n)`, определяющую количество нулей среди всех цифр числа N.

(Естественно, оформите решение в виде программы, использующей эту функцию.)

Формат входных данных

Задано единственное число N.

Формат выходных данных

Необходимо вывести количество нулей среди всех цифр числа N.

Пример

Входные данные	Выходные данные
50	1

Задача F. Обращение числа

Напишите функцию `int reverse(int n)`, которая переставляет цифры числа в обратном порядке.

(Естественно, оформите решение в виде программы, использующей эту функцию.)

Формат входных данных

Задано единственное число **N**.

Формат выходных данных

Необходимо вывести цифры данного числа в обратном порядке.

Пример

Входные данные	Выходные данные
179	971

Задача G. Минимальный делитель

Дано целое число, не меньшее 2. Выведите его наименьший натуральный делитель, отличный от 1.

Пример

Входные данные	Выходные данные
15	3

Задача H. Почти простые числа

Назовем натуральное число *почти простым*, если оно раскладывается на произведение каких-нибудь двух неравных простых.

Входные данные

Натуральное число k до 2 млрд включительно.

Выходные данные

YES, если число k почти простое, и NO, если это не так.

Примеры

Входные данные	Входные данные
6	YES
17	NO
16	NO

Задача I. Полярные единички

Программист на Северном полюсе работал за компьютером в варежках и поэтому мог набирать только 0 и 1, а клавиша 0 запала. Сможет ли он набрать число, состоящее только из единиц и при этом кратное заданному *N*?

Формат входного файла

Программе дано число N ($1 \le N \le 10^6$).

Формат выходного файла

Вывести минимальное число, удовлетворяющее требованию, или "NO" , если такого числа не существует.

Примеры

Входные данные	Входные данные
100	NO
57	11111111111111111

Обратите внимание: количество единичек в числе может получиться очень большим!

Задача J. Разложение на простые++

Требуется разложить целое число *N* на простые множители и вывести результат в порядке возрастания, соблюдая формат, как показано в примере. Крышечка «^» означает степень.

Формат входного файла

Программе дано число N ($2 \le N \le 109$).

Формат выходного файла

Вывести разложение *N* на простые множители.

Входные данные	Входные данные
2	2
1008	2^4*3^2*7

Подсказки и решения. 3 Занятие.

3.1

Свойства сравнений (арифметика остатков)

1. Рефлексивность: $a \equiv a \pmod m$.

Действительно, a – a = 0, а 0 делится на m.

2. Симметричность: Если $a \equiv b \pmod m$, то $b \equiv a \pmod m$.

В самом деле, если a-b делится на m, то и b-a делится на m.

3. Свойство транзитивности. Если $a \equiv b \pmod m$, $b \equiv c \pmod m \Rightarrow a \equiv c \pmod m$.

Доказательство: $a \equiv b \pmod m \Rightarrow a = k \cdot m + b$ (1) и $b \equiv c \pmod m \Rightarrow b = l \cdot m + c$ (2).

Внесём (2) в (1) и получим $a = k \cdot m + l \cdot m + c = m(k+l) + c \Rightarrow a \equiv c \pmod m$.

Это значит, что в правой части сравнений могут стоять только остатки от деления a на m. Например: $60 \equiv 4 \pmod 7$, 60:7=8(остаток 4).

Пример. $27 \equiv 15 \pmod 6$, а $15 \equiv 3 \pmod 6 \Rightarrow 27 \equiv 3 \pmod 6$.

4. Свойство сложения. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$ то $a+c \equiv b+d \pmod m$.

Доказательство: $a = m \cdot k = b$ и $c = m \cdot l + d \Rightarrow a + c = m(k+l) + (b+d) \Rightarrow a + c \equiv b + d \pmod m$.

Пример. $5 \equiv 2 \pmod 3$ и $7 \equiv 4 \pmod 3 \Rightarrow 5+7 \equiv 2+4 \pmod 3 \Rightarrow 12 \equiv 6 \pmod 3$, остаток 0.

5. Свойство вычитания. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то $a-c \equiv b-d \pmod m$.

Доказательство: $a = m \cdot k + b$ и $c = m \cdot l + d \Rightarrow a - c = m(k-l) + (b-d) \Rightarrow a - c \equiv b - d \pmod m$.

Пример. $37 \equiv 17 \pmod 5$ и $27 \equiv 12 \pmod 5 \Rightarrow 37-27 \equiv 17-12 \pmod 5 \Rightarrow 10 \equiv 5 \pmod 5$.

6. Свойство произведения. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$ то $ac \equiv bd \pmod m$.

Доказательство: $a = m \cdot k = b$ и $c = m \cdot l + d \Rightarrow ac = m(mkl + kd + bl) + bd \Rightarrow ac \equiv bd \pmod m$.

Пример. $5 \equiv 9 \pmod 4$ и $7 \equiv 11 \pmod 4 \Rightarrow 5 \cdot 7 \equiv 9 \cdot 11 \pmod 4 \Rightarrow 35 \equiv 99 \pmod 4$.

7. Свойство степени. Если $a \equiv b \pmod m$, то $a^n \equiv b^n \pmod m$.

Доказательство: $a \equiv b \pmod m \Rightarrow a = k \cdot m + b \Rightarrow a^n = (m \cdot k + b)^n = m(\ldots) + b^n \Rightarrow$

$a^n \equiv b^n \pmod m$.

Пример: $7 \equiv 4 \pmod 3 \Rightarrow 7^2 \equiv 4^2 \pmod 3 \Rightarrow 49 \equiv 16 \pmod 3$.

8. Если $a \equiv b \pmod m$,то $a+n \equiv b+n \pmod m$.

Доказательство: $a \equiv b \pmod m \Rightarrow a=k \cdot m+b \Rightarrow a+n=m \cdot k+(b+n) \Rightarrow a+n \equiv b+n \pmod m$.

Пример: $9 \equiv 1 \pmod 8 \Rightarrow 9+3 \equiv 1+3 \pmod 8 \Rightarrow 12 \equiv 4 \pmod 8$.

9. Если $a \equiv b \pmod m$,то $an \equiv bn \pmod m$.

Доказательство: $a \equiv b \pmod m \Rightarrow a=k \cdot m+b \Rightarrow a \cdot n=m \cdot k \cdot n+b \cdot \Rightarrow a \cdot n \equiv b \cdot n \pmod m$.

Пример: $27 \equiv 15 \pmod 6 \Rightarrow 27 \cdot 3 \equiv 15 \cdot 3 \pmod 6 \Rightarrow 81 \equiv 45 \pmod 6$.

10. Если $a \equiv b \pmod m$, то $a/\alpha \equiv b/\alpha \pmod{m/\alpha}$, если α общий делитель a, b, и m.

Доказательство: $a/\alpha \equiv b/\alpha \pmod{m/\alpha} \Rightarrow a/\alpha=m/\alpha \cdot k+b/\alpha \Rightarrow a/\alpha \equiv b/\alpha \pmod{m/\alpha}$.

Пример: $56 \equiv 8 \pmod 8 \Rightarrow 56/4 \equiv 8/4 \pmod{12/4} \Rightarrow 14 \equiv 2 \pmod 3$.

11. Если $a \equiv b \pmod m$, то $a/\alpha \equiv b/\alpha \pmod m$, если α общий делитель a и b, но не делитель m.

Доказательство: $a \equiv b \pmod m \Rightarrow a=k \cdot m+b \Rightarrow a-b=m \cdot k$ (1), a делится на $\alpha \Rightarrow a=\alpha x$, b делится на $\alpha \Rightarrow b=\alpha y$, подставим в (1), $\alpha x-\alpha y=mk \Rightarrow \alpha(x-y)=m \cdot k$ (2),

$\alpha(x-y)$ делится на $\alpha \Rightarrow k$ делится на α (m не делится на α!) $\Rightarrow k=\phi \cdot \alpha (\phi \in N)$ подставим в (2), $\alpha(x-y)=m \cdot \alpha \cdot \phi \Rightarrow x-y=m \cdot \phi \Rightarrow x \equiv y \pmod m$, но $x=a/\alpha$ и $y=b/\alpha \Rightarrow a/\alpha \equiv b/\alpha \pmod m$.

Пример: $9 \equiv 3 \pmod 2 \Rightarrow 9/3 \equiv 3/3 \pmod 2 \Rightarrow 3 \equiv 1 \pmod 2$.

Таким образом, со сравнениями можно поступать, как и с равенствами: почленно складывать, вычитать, умножать и возводить в натуральную степень; члены сравнений можно переносить в другую часть с противоположным знаком, умножать на любое натуральное число и делить на общий делитель с учётом свойства 9.

3.2

Заметим, что $\underbrace{111\ldots1}_{N+1 \text{ единица}} = \underbrace{11\ldots1}_{N \text{ единиц}} * 10 + 1$

Тогда можно вычислять остатки последовательно: `m = (m * 10 + 1) % p;`

3.3

Последняя цифра числа это остаток от деления его на 10. Поэтому найти ее можно операцией «%»: `x % 10`

3.4

Пока x > 0. Ведь деление происходит нацело, и рано или поздно это произойдет. Нужно отдельно рассматривать случай, когда переменная `x` равна 0 изначально.

3.5

Для двух функция работает правильно. Мы даже не войдем в цикл! А для единицы возвратит true. Исправить это можно прямо в возврате:

```
return result && N != 1;
```

3.6

Из основной теоремы арифметики следует, что если число раскладывается в произведение двух простых, то только их мы и найдем.

3.7

Кажется, что в задаче ничего не дано. Просто вообще ничего. Однако условие позволяет дать абсолютно точный ответ. На помощь приходят остатки!

Пусть было n овец в стаде. Тогда пастухи выручили n^2 тугриков. Причем число десятков нечетно.

Так как $n^2 = (10a + b)^2 = 100a^2 + 20ab + b^2$, то число десятков должно быть нечётным в b^2. Этому условию удовлетворяют лишь две цифры: 4 и 6 (их квадраты равны соответственно 16 и 36). Оба квадрата оканчиваются цифрой 6, следовательно, число n^2 также оканчивается цифрой 6. Таким образом, после обмена денег в банке у пастухов было 6 монет. А значит, ножик стоил 2 тугрика.

Разбор задач . Занятие 3

3A. Уравнение

Уравнение ***AX + B = 0*** имеет

- бесконечно много решений в целых числах при A и B равных 0,
- не имеет решений, если A = 0, а B нет
- если A ≠ 0, то A должно делиться на B, чтобы оно имело единственный корень.

Если A и B равны нулю – ответ «INF»

```
if (A == 0 && B == 0) out.print("INF");
```

Если A ≠ 0 и B делится на A – частное с противоположным знаком:

```
else if (A != 0 && B % A == 0) out.print(-B / A);
```

иначе «NO»:

```
else out.print("NO");
```

3B. Фишки

В данной задаче фишки расставлены по периметру клетчатой квадратной доски. . Зная формулу для нахождения периметра клетчатой доски: $P = 4a - 4$, где P - периметр; a - сторона квадрата, получим, что $a = \frac{P-4}{4}$. Чтобы задача имела решение, необходимо, чтобы a имело смысл. Таким образом, в задаче нужно *проверить делится ли введённое число P на 4* (в этом случае и a будет делиться на 4 по свойству разности сравнений).

Также нужно отдельно рассмотреть случаи, когда введённое число P равно 0 и 1.

3C. Мороженое

Конечно, можно использовать циклы, пытаясь набрать требуемое количество тройками и пятерками. Но задача решается просто, если понять, что тройками и пятерками можно набрать любое число большее семи! Это несложно доказать. Разделим данное число на 3 с остатком. Если получился остаток 0 – все готово. Если остаток 1 - заменим три тройки двумя пятерками. Если 2 – две тройки одной пятеркой.

Таким образом, *ответ «NO» получается только в случаях 1, 2, 4 и 7 шариков. В остальных случаях ответ «YES»*.

3D. Автобусы

Во первых, нужно учесть, что K может принимать значение меньше или равное 2. В этом случае следует выводить 0, потому, что в каждый автобус мы будем вынуждены посадить взрослых (а дети так и не уедут).

Теперь рассмотрим случай когда *K* больше двух: в этом случае нужно будет *N/(K-2)* автобусов для перевозки детей. Заметим, что если *N* не делится нацело на *K-2*, то автобусов понадобится на один больше.

Так же мы не сможем уехать, если количество взрослых, делённое на два, меньше количества нужных автобусов для перевозки детей.

Во всех остальных случаях ответом будет *(M+N)/K*, но если *M+N* не делится нацело на *K*, то на один автобус больше:

```
// ...выше рассматриваются все особенные «NO» случаи
else
{
     res = (M + N) / K;
     if ((M + N) % K != 0) res++;
     out.print(res);
}
```

3E. Количество нулей

Просто добавим в рассмотренный в тексте занятия цикл увеличение счетчика, если очередная цифра – ноль:

```
...
c = x % 10;
// В переменной c очередная цифра -
// делаем что-нибудь с ней...
if (c == 0) nZeros++;
```

3F. Обращение числа

Будем использовать цикл получения цифр, как описано в занятии. При получении очередной цифры числа будем добавлять ее справа к новому числу.

```
До цикла заводим переменную res
// int res = 0;
// ...
// В переменной c очередная цифра -
// прибавляем ее справа к res
res = res * 10 + c;
// ...
```

3G. Минимальный делитель

Будем в цикле последовательно перебирать все делители, начиная с двойки. Как только найдем – выйдем из цикла:

```
// ... d «пробегает» в цикле значения от 2 до N
if (N % d == 0) break;
```

и в конце напечатаем d.

Заметим, что правильное d будет найдено в любом случае, ведь любое число делится хотя бы на само себя.

Эта программа будет работать долго, если дано простое число – поиск пройдет до него. Можно это оптимизировать так, как разбирается в тексте занятия - перебирать возможные делители до корня из N, и если не нашли – сразу выводить N. Однако тесты к задаче позволяют получить «ОК» и без такой оптимизации.

3H. Почти простые числа

Запустим цикл по поиску нетривиальных делителей, который разбирается в тексте занятия. В нем будем считать количество найденных делителей (Nd).

```
Nd++;
if (N / d != d) Nd++; // если парный делитель не совпадает с делителем
```

Из основной теоремы арифметики следует, что если число раскладывается в произведение двух простых, то только их мы и найдем. Если насчитали 2 – «YES», иначе – «NO».

3I. Полярные единички

Заметим, что $\underbrace{111\ldots1}_{\text{N+1 единица}} = \underbrace{11\ldots1}_{\text{N единиц}} * 10 + 1$

Тогда можно вычислять остатки последовательно:

В начале

`m = 1` (остаток от деления единицы на N),

`e = 1` (количество единичек).

Дальше в цикле

`m = (m * 10 + 1) % N` . и будем считать количество добавленных единичек (`e++`).

Когда заканчивать цикл? Как только m станет равно нулю. А если это никогда не произойдет? Но остатков конечное число – всего p. Сделаем операцию p раз. Если мы ни

разу не встретили ноль, значит, серии остатков начали повторяться! В этом случае нужно вывести ответ «NO». Если же мы получили ноль – еще в одном цикле просто выводим `e` единичек.

3J. Разложение на простые++

Это технически сложная задача. В цикле разложения на простые нужно накапливать степень и выводить ее вместе со знаком «^». Нужно учесть, что первая степень не выводится. И после последнего множителя не ставится знак умножения. Ограничения задачи вынуждают оптимизировать цикл в случае больших простых.

Занятие 3.Справочник

Проверка на делимость, четность

```
if (x % k == 0)
{
     // число x делится на k
}
```

в частности проверка на четность:

```
if (x % 2 == 0)
{
     // число x - четное
}
```

Получение цифр числа справа налево

```
while(x > 0)
{
     c = x % 10;
     // В переменной c очередная цифра -
     // делаем что-нибудь с ней...
     x /= 10; // уменьшаем переменную x в 10 раз
}
```

Нужно отдельно рассматривать случай, когда в переменой x ноль изначально.

Проверка на простоту

```
boolean isPrime(int N)
{
     int d = 2; // начинаем проверять с двух
     boolean result = true; // "презумпция" простоты
     while (d <= N / d) // парный делитель к d равен N / d
     {
          if (N % d == 0) // d - делитель
          {
               result = false; // составное!
               break; // немедленно выходим из цикла
          }
          d++;
     }
     return isPrime && N != 1;
}
```

Разложение на простые множители

```
int d = 2; // начинаем проверять с двух
while ( N > 1) // пока не уменьшим до 1
{
     if (N % d == 0) // d – делитель
     {
          out.print(d + " ");
          N /= d;
     }
     d++;
}
```

Важно! Этот код медленно работает для простых чисел.