Февраль 27, 2022

9098. XOR сумма

Вычислите XOR всех чисел на отрезке $[l, r]$.

Вход. Два целых числа l и r ($0 \le l \le r \le 10^{18}$).

Выход. Выведите значение XOR всех чисел на отрезке $[l, r]$.

Пример входа	**Пример выхода**
3 8	11

XOR четного и следующего за ним нечетного числа равен 1. Следовательно XOR четырех последовательных чисел, начиная с четного, равен 0. Таким образом XOR на промежутке $[4a, 4a + 1, \dots 4b + 2, 4b + 3]$, где $a \le b$, равен 0.

Разобьём интервал $[l, r]$ на три:

$$[l, r] = [l, 4a - 1] \cup [4a, 4a + 1, \dots 4b + 2, 4b + 3] \cup [4b + 4, r],$$

где

- $4a$ – наименьшее число, не меньшее l, делящееся на 4,
- $4b + 3$ – наибольшее число, не большее r, при делении на 4 дающее остаток 3

XOR среднего интервала равен 0. XOR первого и последнего интервала вычисляем непосредственно.

Второе решение. XOR на промежутке $[l, r]$ может быть вычислен как XOR на промежутке $[0, r] \oplus$ XOR на промежутке $[0, l - 1]$ или то же самое что

$$\mathrm{XOR}([l, r]) = \mathrm{XOR}([0, r]) \oplus \mathrm{XOR}([0, l - 1])$$

Пусть f(n) = XOR([0, n]). Тогда

- f(n) = 0, если $n \le 0$;
- f(n) = n, если n делится на 4;
- f(n) = 1, если при делении n на 4 получается остаток 1;
- f(n) = n XOR 1, если при делении n на 4 получается остаток 2;
- f(n) = 0, если при делении n на 4 получается остаток 3;

Пример. Рассмотрим интервал [2; 12] = [2; 3] U [4; 11] U [12]. Тогда

XOR([2; 12]) = XOR([2; 3]) $\oplus$ XOR([4; 11]) $\oplus$ XOR([12]) =

$2 \oplus 3 \oplus 0 \oplus 12 = 0010_2 \oplus 0011_2 \oplus 0000_2 \oplus 1100_2 = 1101_2 = 13$

Второй способ

XOR([2; 12]) = XOR([0; 12]) $\oplus$ XOR([0; 1]) = 12 $\oplus$ 1 = 13

Реализация алгоритма

Читаем входные данные. В переменной *res* подсчитываем искомый XOR.

```
scanf("%lld %lld", &l, &r);
res = 0;
```

Двигаем левый конец *l* вперед, пока он не достигнет числа, делящегося на 4.

```
while (l % 4 != 0 && l <= r)
{
  res ^= l;
  l++;
}
```

Двигаем правй конец *r* назад, пока он не достигнет числа, которое приделении на 4 дает остаток 3.

```
while (r % 4 != 3 && l <= r)
{
  res ^= r;
  r--;
}
```

Выводим ответ.

```
printf("%lld\n", res);
```

Реализация алгоритма – XOR([*l*, *r*]) = XOR([0, *r*]) $\oplus$ XOR([0, *l* – 1])

```
#include <stdio.h>

long long i, l, r, res;

long long f(long long n)
{
  if (n <= 0) return 0;
  if (n % 4 == 0) return n;
```

```
  if (n % 4 == 1) return 1;
  if (n % 4 == 2) return n ^ 1;
  return 0;
}

int main()
{
  scanf("%lld %lld", &l, &r);
  res = f(r) ^ f(l - 1);
  printf("%lld\n", res);
}
```

936. Формулы Крамера

Решить систему двух линейных уравнений с двумя неизвестными по формулам Крамера. Система уравнений, приведенная во входных данных, имеет вид:

$$\begin{cases} 5x_1 + 8x_2 = 11 \\ -3x_1 + 6x_2 = 15 \end{cases}$$

Вход. Первая строка содержит коэффициенты первого уравнения, а вторая строка содержит коэффициенты второго. Все входные числа разделены одним пробелом и не превышают 100 по модулю.

Выход. Первый корень системы уравнений вывести в первой строке, а второй корень во второй строке с точностью до 0.001.

Пример входа	**Пример выхода**
5 8 11 -3 6 15	-1.000 2.000

Определителем называется число, равное

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = a * d - b * c$$

Рассмотрим систему уравнений:

$$\begin{cases} a_1 x + b_1 y = c_1 \\ a_2 x + b_2 y = c_2 \end{cases}$$

Помножим первое уравнение на b_2, а второе на b_1:

$$\begin{cases} a_1 b_2 x + b_1 b_2 y = c_1 b_2 \\ a_2 b_1 x + b_1 b_2 y = c_2 b_1 \end{cases}$$

Вычтем из первого уравнения второе:

$$(a_1 b_2 - a_2 b_1)\, x = c_1 b_2 - c_2 b_1$$

Или то же самое что

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} x = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}$$

Аналогично в исходной системе первое уравнение умножим на a_2, а второе на a_1:

$$\begin{cases} a_1 a_2 x + a_2 b_1 y = a_2 c_1 \\ a_1 a_2 x + a_1 b_2 y = a_1 c_2 \end{cases}$$

Вычтем из второго уравнения первое:

$$(a_1 b_2 - a_2 b_1)\, y = a_1 c_2 - a_2 c_1$$

Или то же самое что

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$

Обозначим

$$d = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix},\ d_x = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix},\ d_y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$

Тогда решение системы уравнений можно записать в виде:

$$x = \frac{d_x}{d},\ y = \frac{d_y}{d},\ d \neq 0$$

Пример. Рассмотрим систему уравнений $\begin{cases} 5x_1 + 8x_2 = 11 \\ -3x_1 + 6x_2 = 15 \end{cases}$. Для нее имеем:

$$d = \begin{vmatrix} 5 & 8 \\ -3 & 6 \end{vmatrix} = 5*6 + 3*8 = 54,$$

$$d_x = \begin{vmatrix} 11 & 8 \\ 15 & 6 \end{vmatrix} = 11*6 - 15*8 = -54,\ d_y = \begin{vmatrix} 5 & 11 \\ -3 & 15 \end{vmatrix} = 5*15 + 3*11 = 108,$$

откуда

$$x = -54 / 54 = -1,\ y = 108 / 54 = 2$$

Реализация алгоритма

Функция ***kramer*** решает систему линейных уравнений методом Крамера. Корни системы возвращаются в переменных *x* и *y*.

```
int kramer(double a1, double b1, double c1,
           double a2, double b2, double c2, double &x, double &y)
{
  double d = a1 * b2 - a2 * b1;
  double dx = c1 * b2 - c2 * b1;
  double dy = a1 * c2 - a2 * c1;
```

При $d = 0$ прямые параллельны.

- Если $dx = 0$ (при этом будет и $dy = 0$), то прямые совпадают, возвращаем 2.
- Если $dx \neq 0$ (при этом будет и $dy \neq 0$), то прямые не совпадают (не имеют общих точек), возвращаем 1.

```
  if (d == 0) return (dx == 0.0) + 1;
```

При $d \neq 0$ система имеет единственное решение, которое вычисляем в паре (x, y). В этом случае возвращаем 0.

```
  x = dx / d; y = dy / d;
  return 0;
}
```

Основная часть программы. Читаем входные данные.

```
scanf("%lf %lf %lf",&a1,&b1,&c1);
scanf("%lf %lf %lf",&a2,&b2,&c2);
```

Решаем систему уравнений и выводим ответ.

```
kramer(a1,b1,c1,a2,b2,c2,x,y);
printf("%.3lf\n%.3lf\n",x,y);
```

2164. Шифр Юлия

Юлий Цезарь использовал свой способ шифрования текста. Каждая буква заменялась на следующую по алфавиту через k позиций по кругу. Необходимо по заданной шифровке определить исходный текст.

Вход. В первой строке дана шифровка, состоящая из не более чем 255 заглавных латинских букв. Вторая строка содержит число k ($1 \le k \le 10$).

Выход. Вывести результат расшифровки.

Пример входа	**Пример выхода**
XPSE 1	WORD

Для расшифровки следует каждую букву прокрутить на k позиций назад по кругу. Пронумеруем буквы от 0 (‘A’) до 25 (‘Z’).

Пусть s[i] – текущая буква шифра. Тогда номер буквы равен s[i] – ‘A’. Вычтем из нее число k по кругу (перед ‘A’ идет ‘Z’), получим (s[i] – ‘A’ – k + 26) % 26. Осталось прибавить к ней ‘A’, получив ASCII код. Таким образом операция дешифрования имеет вид:

$$s[i] = (s[i] - \text{‘A’} - k + 26) \% 26 + \text{‘A’}$$

Реализация алгоритма

Входную строку содержим в символьном массиве s.

```
char s[1000];
```

Читаем входные данные.

```
gets(s); scanf("%d", &k);
```

Пересчитываем символы строки согласно шифру.

```
for (i = 0; i < strlen(s); i++)
  s[i] = (((s[i] - 'A') + 26 - k) % 26) + 'A';
```

Выводим результат расшифровки.

```
puts(s);
```

1417. Ход ферзя

На доске $m \times n$ стоит ферзь. Определите, сколько клеток находится под боем ферзя.

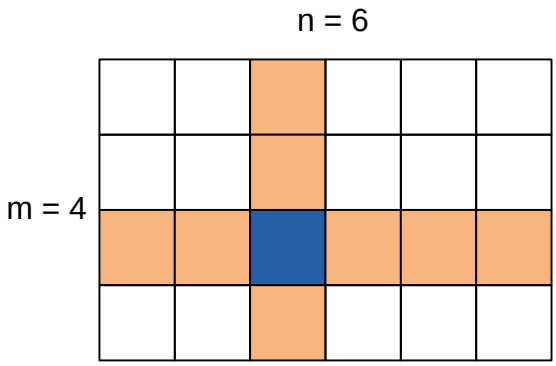
Вход. Четыре натуральных числа: размеры доски m и n и координаты ферзя x и y ($1 \le x \le m \le 10^9$, $1 \le y \le n \le 10^9$).

Выход. Вывести количество клеток под боем ферзя.

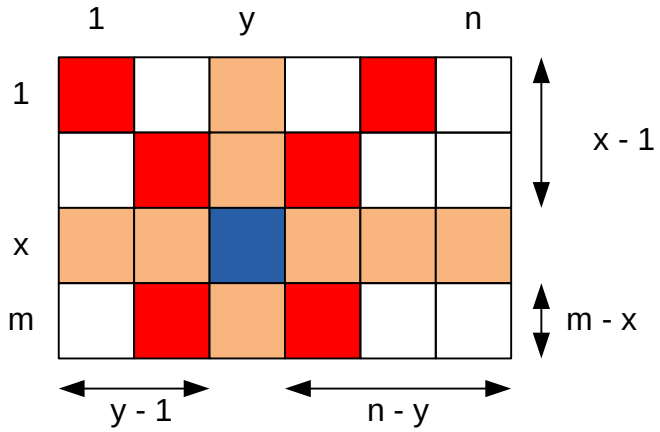
Пример входа	**Пример выхода**
8 8 4 5	27

Ферзь – это шахматная фигура, которая ходит (и бьет) по горизонтали и вертикали.

Строка, в которой стоит ферзь, содержит n клеток. Следовательно количество клеток, на которое ферзь может пойти горизонтальным ходом, равно n – 1. Столбец, в котором стоит ферзь, содержит m клеток. Следовательно количество клеток, на которое ферзь может пойти вертикальными ходом, равно m – 1. То есть под горизонтальным и вертикальным ударом ферзя находится n – 1 + m – 1 клеток.



Разобьем доску на 4 части горизонтальной и вертикальной прямой, проходящей через клетку с ферзем. В каждой такой части подсчитаем количество клеток под ударом.



Рассмотрим например правую верхнюю четверть. Ее длина составляет $n - y$ клеток, а ширина $x - 1$ клеток. Следовательно клеток под боем ферзя будет $\min(x - 1, n - y)$. Аналогично для:

- левой верхней четверти под боем ферзя будет $\min(x - 1, y - 1)$ клеток;
- левой нижней четверти под боем ферзя будет $\min(m - x, y - 1)$ клеток;
- правой нижней четверти под боем ферзя будет $\min(m - x, n - y)$ клеток;

Реализация алгоритма

Читаем входные данные.

```
scanf("%d %d %d %d", &m, &n, &x, &y);
```

Заносим в переменную s количество клеток, которое бьет ферзь горизонтальными и вертикальными ходами.

```
s = m - 1 + n - 1; // horiz & vertical
```

Добавляем диагональные клетки, находящиеся под ударом.

```
s += min(m - x, n - y); // (x,y) = (m, n)
s += min(m - x, y - 1); // (x,y) = (m, 1)
s += min(x - 1, n - y); // (x,y) = (1, n)
s += min(x - 1, y - 1); // (x,y) = (1, 1)
```

Выводим ответ.

```
printf("%lld\n",s);
```

1249. Первая цифра степени

Вам предлагается очень простая задача: на какую цифру начинается число n^n?

Вход. Состоит из нескольких тестов. Каждый тест расположен в отдельной строке и содержит единственное число n ($1 \le n \le 10^9$).

Выход. Для каждого теста вывести в отдельной строке первую цифру искомого результата.

Пример входа	**Пример выхода**
3	2
4	2

Количество цифр числа a равно $\lfloor \lg a \rfloor + 1$. Следовательно число n^n содержит $len = \lfloor n \lg n \rfloor + 1$ десятичных цифр. Рассмотрим число $A = \frac{n^n}{10^{len}} \cdot 10 = \frac{10^{n \lg n}}{10^{len}} \cdot 10 = 10^{n \lg n + 1 - len}$. Очевидно, что $1 \le A < 10$. При этом целая часть числа A равна первой цифре числа n^n.

Вычисляем десятичный логарифм числа A. Он равен $ost = n\lg n + 1 - len$. Тогда первая цифра числа n^n равна $\lfloor 10^{ost} \rfloor$.

Пример. Пусть $n = 4$. Тогда $4^4 = 256$, $len = \lfloor 4 \lg n 4 \rfloor + 1 = 3$. Число A равно $\frac{4^4}{10^3} \cdot 10 = 10^{4 \lg 4 + 1 - 3} = 10^{0.40824} = 2.56$.

Реализация алгоритма

Основной цикл программы.

```
while(scanf("%lld",&n) == 1)
{
  lgg = n * log10((double)n);
```

Значение *len* равно количеству цифр в числе n^n.

```
  len = (long long)(lgg + 1e-7) + 1;
  ost = lgg + 1 - len;
  res = (long long)(pow(10.0,ost) + 1e-7);
  printf("%lld\n",res);
}
```

4213. Города

Юный программист решил придумать собственную игру. Игра происходит на поле размером $n \times n$ клеток, в некоторых клетках которого расположены города (каждый город занимает одну клетку; в каждой клетке может располагаться не более одного города). Всего должно быть чётное количество городов.

Изначально про каждую клетку игрового поля известно, расположен ли в ней город или нет. Чтобы начать игру, необходимо разделить игровое поле на два государства так, чтобы в каждом государстве было поровну клеток-городов.

Граница между государствами должна проходить по границам клеток таким образом, чтобы из любой клетки каждого государства существовал путь по клеткам этого же государства в любую другую его клетку (из клетки можно перейти в соседнюю, если они имеют общую сторону). Каждая клетка игрового поля должна принадлежать только одному из двух государств, при этом государства не обязаны состоять из одинакового количества клеток.

Напишите программу, которая разделит клетки заданного игрового поля между двумя государствами.

Вход. Первая строка содержит размер игрового поля n ($1 \le n \le 50$).

Последующие n строк содержат по n заглавных латинских букв (без пробелов), кодирующих соответствующие клетки игрового поля: C обозначает клетку, занятую городом, D – пустую клетку. Гарантируется, что на поле есть хотя бы два города и всего их чётное число.

Выход. Вывести n строк по n цифр (без пробелов) в каждой, кодирующих соответствующие клетки. Цифра 1 обозначает, что данная клетка принадлежит первому государству, цифра 2 – данная клетка принадлежит второму государству.

Если решений несколько, необходимо вывести любое из них.

Пример входа 1

```
3
DDD
DDC
DDC
```

Пример выхода 1

```
222
212
211
```

Пример входа 2

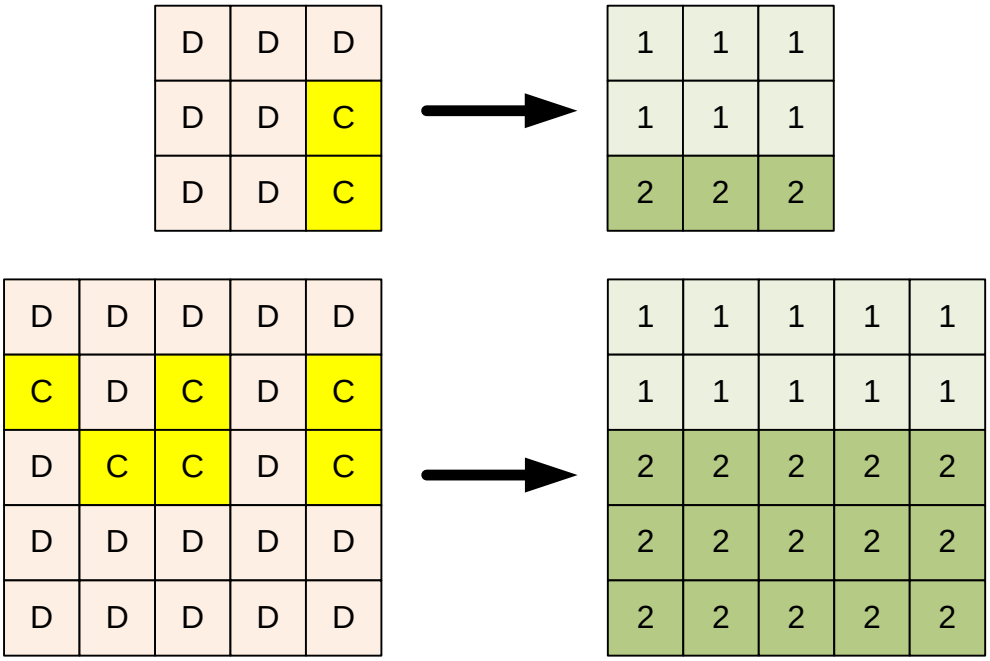
```
5
DDDDD
CDCDC
DCCDC
DDDDD
DDDDD
```

Пример выхода 2

```
11111
12221
12221
11111
```

Вычислим количество городов *cnt* на игровом поле. Первые *cnt* / 2 городов относим к первому государству, остальные – ко второму. Пока все *cnt* / 2 городов не будут отнесены к первому государству, пустые клетки будем также относить к первому государству. При таком разделении обе области государств будут связными.

Пример



Реализация алгоритма

Входное игровое поле читаем в массив s. Результирующее поле строим в массиве res.

```
#define MAX 55
char s[MAX][MAX], res[MAX][MAX];
```

Читаем входные данные.

```
scanf("%d\n",&n);
for(i = 0; i < n; i++)
  gets(s[i]);
```

В переменной *cnt* подсчитаем количество городов.

```
cnt = 0;
for(i = 0; i < n; i++)
for(j = 0; j < n; j++)
  if (s[i][j] == 'C') cnt++;
```

Двигаемся по полю сверху вниз и слева направо. Первые *cnt* / 2 городов относим к первому государству, остальные – ко второму. Пока все *cnt* / 2 городов не будут отнесены к первому государству, пустые клетки будем также относить к первому государству.

```
cnt /= 2;
for(i = 0; i < n; i++)
for(j = 0; j < n; j++)
{
  if (cnt > 0) res[i][j] = '1'; else res[i][j] = '2';
  if (s[i][j] == 'C') cnt--;
}
```

Выводим разделенное игровое поле между двумя государствами.

```
for(i = 0; i < n; i++)
  puts(res[i]);
```

7792. Обычный мальчик

Петя – обычный мальчик. Каждый раз перед началом каждого учебного года у него появляется желание взяться за ум. И в этом году он не отступил от своей цели сразу после начала учебы. Внимательно слушая учителя и выполняя все домашние задания, он стал лучшим учеником в классе.

Учитель заметил успехи Пети и выдал ему самый сложный вариант контрольной по математике. Почти все задания мальчик решил за урок, но с одной так и не справился.

Дано целое положительное число x. Требуется найти число $y \geq x$, у которого не менее 100 делителей. Причем число y должно превышать число x не более чем на 1%, то есть должно выполняться неравенство $x \leq y \leq 1.01x$.

Помогите Пете решить эту сложную задачу.

Известно, число 510510 имеет ровно 128 делителей.

Вход. Первая строка ввода содержит число x ($1 \leq x \leq 10^{16}$).

Выход. Если подходящего числа не существует, то выведите -1, иначе выведите любое подходящие число.

Пример входа	**Пример выхода**
510000	510510

Попробуем найти некоторое наименьшее по возможности число k, которое содержит 100 делителей. Пусть оно имеет вид $2^a3^b5^c7^d$. Тогда количество его делителей $(a + 1) * (b + 1) * (c + 1) * (d + 1)$ должно превышать 100. Выберем например $a = 5$, $b = 2$, $c = 2$, $d = 1$ ($6 * 3 * 3 * 2 = 108 > 100$). Тогда $k = 2^5 3^2 5^2 7^1 = 32 * 9 * 25 * 7 = 50400$.

Пусть $x \geq 100k = 5\ 040\ 000$. Рассмотрим число $(x / 50\ 400 + 1) * 50\ 400 \leq x + 50\ 400 = 100k + k = 101k$. То есть число $(x / 50\ 400 + 1) * 50\ 400$ делится на 50400 и поэтому имеет больше 100 делителей, и при этом не больше $101k$ (то есть не превосходит x на 1%). Следовательно при $x \geq 5\ 040\ 000$ в качестве ответа можно вывести число $(x / 50\ 400 + 1) * 50\ 400$.

В противном случае совершим полный перебор. Для заданного x переберем все числа i от x до $1.01x$, для каждого такого i вычислим количество делителей. Если для некоторого i это количество не менее 100, то выводим в качестве ответа число i. Иначе подходящего числа не существует.

Реализация алгоритма

Функция Div100 вычисляет количество делителей числа n. Если оно больше или равно 100, возвращаем 1. Иначе возвращаем 0.

```
int Div100(int n)
{
  int c, i, res = 1;
  for(i = 2; i <= (int)sqrt(1.0*n); i++)
  {
    if (n % i == 0)
    {
      c = 0;
      while(n % i == 0)
      {
        n /= i; c++;
      }
      res *= (c + 1);
    }
  }
  if (n > 1) res *= 2;
  return res >= 100;
}
```

Основная часть программы. Читаем входные данные.

```
scanf("%lld",&x);
```

Если $x \geq 5\,040\,000$, то выводим $(x / 50\,400 + 1) * 50\,400$.

```
if (x >= 5040000)
{
  printf("%lld\n",(x / 50400 + 1) * 50400);
  return 0;
}
```

Перебираем i от x до $101 * x / 100$. Если количество делителей у числа i не меньше 100, то выводим его.

```
for(i = x; 100 * i <= 101 * x; i++)
  if (Div100(i))
  {
    printf("%d\n",i);
    return 0;
  }
```

Если требуемое число не найдено, то выводим -1.

```
printf("-1\n");
```

4076. Генератор простых чисел

Петр хочет сгенерировать несколько простых чисел для своей криптосистемы. Помогите ему! Вам следует сгенерировать все простые числа между двумя заданными.

Вход. В первой строке содержится количество тестов t ($t \le 10$). В каждой из следующих t строк содержится два числа m и n ($1 \le m \le n \le 10^9$, $n - m \le 100000$), разделенных одним пробелом.

Выход. Для каждого теста вывести все простые числа p, удовлетворяющие $m \le p \le n$, по одному числу в строке. Тесты следует разделять пустой строкой.

Пример входа

```
2
1 10
3 5
```

Пример выхода

```
2
3
5
7

3
5
```

В задаче необходимо реализовать решето Эратосфена на отрезке. Информацию о простоте чисел [m … n] будем хранить в primes[0 ... $n - m$]. Перебираем все возможные делители p = 2, 3, 5, 7, …, $\lfloor\sqrt{n}\rfloor$ чисел на отрезке и отмечаем в массиве primes кратные им как составные.

Пусть *low* – наименьшее число из отрезка [m … n], кратное p. Тогда все числа $j = low + pk$ (k = 0, 1, 2, …), не большие n, следует установить составными (primes[$j - m$] = 0) так как все они делятся на p. При этом если $j = p$, и p простое то primes[$j - m$] следует оставить равным 1 (число j простое).

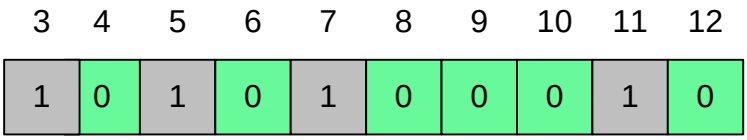
Пример. Промоделируем работу решета на отрезке [3; 12]. Изначально все числа объявим простыми. Поскольку $\lfloor\sqrt{12}\rfloor = 3$, то достаточно перебрать делители 2 и 3.

3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1

p = 2: все числа, кратные 2, объявляем составными. Числа 2 в массиве нет.

3	4	5	6	7	8	9	10	11	12
1	0	1	0	1	0	1	0	1	0

p = 3: все числа, кратные 3, объявляем составными. Число 3 в массиве есть, поэтому его оставляем простым.



Реализация алгоритма

В массиве primes будем отмечать простоту чисел от m до n: ячейка содержит 1, если число простое и 0 иначе. При этом ячейка primes[0] содержит информацию о простоте числа m.

```
int primes[100010];
```

Читаем входные данные.

```
scanf("%d",&tests);
while(tests--)
{
  scanf("%d %d",&m,&n);
```

Изначально все числа на отрезке [m .. n] пусть будут простыми.

```
  for(i = 0; i < n - m + 1; i++) primes[i] = 1;
```

Перебираем все возможные делители p чисел с отрезка: стартуем с двойки, а потом перебираем все нечетные числа, начиная с 3 и вплоть до $\sqrt{n}$.

```
  for(p = 2; p <= (int)sqrt(1.0*n); p += 2)
  {
```

Пусть наименьшее число, которое делится на p и не меньше m, равно *low*.

```
    low = m / p * p;
    if (low < m) low += p;
```

Проходим циклом по j от *low* до n с шагом p. Все пройденные числа j будут делиться на p, поэтому в массиве primes отмечаем их как составные (primes[j – m] = 0). При этом:

- Если j = p простое, то primes[p – m] следует оставить равным 1.
- Если j = p – нечетное составное, которое принадлежит отрезку [m .. n], то primes[p – m] установлено в 0 было ранее, когда рассматривался простой делитель числа p.

```
    for(j = low; j <= n; j += p)
      if (j != p) primes[j - m] = 0;
```

Изначально p равнялось 2. Далее в цикле p будет принимать значения нечетных чисел: 3, 5, 7, ...

```
    if (p == 2) p--;
  }
```

Выводим простые числа на отрезке [*m* .. *n*]. Число 1 не является простым.

```
  for(i = m; i <= n; i++)
    if ((primes[i - m] == 1) && (i != 1)) printf("%d\n",i);
```

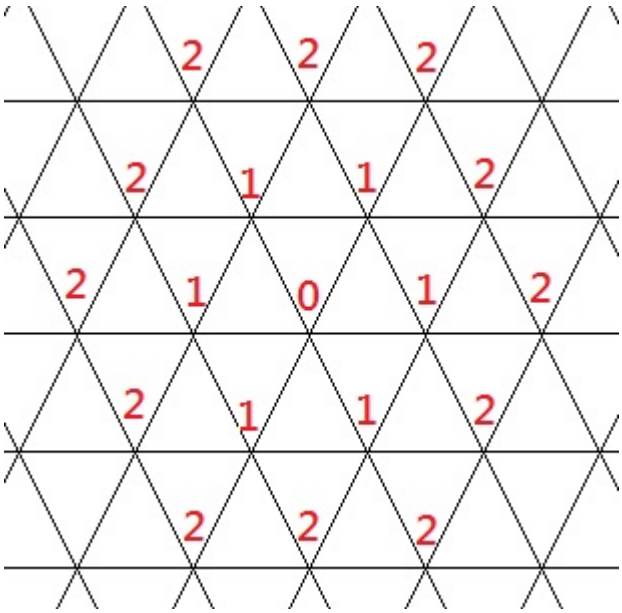
Тесты между собой разделяем пустой строкой.

```
  if (tests) printf("\n");
}
```

4368. Треугольная паутина

Перед Вами бесконечная треугольная сетка. Она устроена таким образом, что если поджечь какую-нибудь вершину, то эта вершина загорается, в следующую секунду загораются все вершины, соседние непосредственно с данной, далее все вершины, соседние с уже горящими, и т.д. Считайте, что огонь никогда не тухнет.

Изначально подожжена одна вершина. Требуется найти количество горящих вершин через *n* секунд.



Вход. Одно число n ($0 \le n \le 10^9$).

Выход. Вывести количество горящих вершин через *n* секунд.

Пример входа 1
1

Пример выхода 1
7

Пример входа 2
1500

Пример выхода 2
6754501

Первый уровень (вершины, которые загораются через 1 секунду) содержит 6 вершин. Второй уровень содержит 12 вершин, третий 18 и так далее. Получается арифметическая прогрессия с разностью 6.

Чрез n секунд будет гореть $res = 1 + (6 + 12 + 18 + \ldots + (6 + 6 * (n - 1)))$ вершин. Используя формулу суммы арифметической прогрессии $\frac{2a_1 + d(n-1)}{2}n$, получим что

$$res = 1 + \frac{2*6 + 6(n-1)}{2}n = 1 + (6 + 3*(n - 1)) * n$$

Реализация алгоритма

Читаем значение n. Вычисляем и выводим ответ.

```
scanf("%lld",&n);
res = 1 + (6 + 3 * (n - 1)) * n;
printf("%lld\n",res);
```

1520. Нечетные делители

Пусть $f(n)$ – наибольший нечетный делитель натурального числа n. По заданному натуральному n необходимо вычислить значение суммы $f(1) + f(2) + ... + f(n)$.

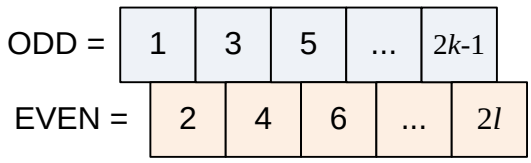
Вход. Каждая строка содержит одно натуральное число n ($n \le 10^9$).

Выход. Для каждого значения n в отдельной строке вывести значение суммы $f(1) + f(2) + ... + f(n)$.

Пример входа	**Пример выхода**
7 1 777	21 1 201537

Если число n нечетное, то $f(n) = n$. Если число n четное, то $f(n) = f(n / 2)$.

Пусть $g(n) = f(1) + f(2) + ... + f(n)$. Разобьем множество натуральных чисел от 1 до n на два подмножества: нечетных ODD и четных EVEN чисел.



Среди натуральных чисел от 1 до n имеется в точности

$k = \lfloor (n+1)/2 \rfloor$ нечетных и $l = \lfloor n/2 \rfloor$ четных чисел

Тогда $f(1) + f(3) + f(5) + ... + f(2k-1) = 1 + 3 + 5 + \ldots + (2k-1) = \frac{1+2k-1}{2} \cdot k = k^2$

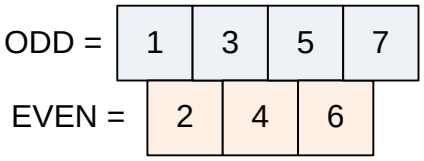
В то же время $f(2) + f(4) + f(6) + ... + f(2l) = f(1) + f(2) + f(3) + ... + f(l) = g(l) = g(\lfloor n/2 \rfloor)$

Таким образом $g(n) = k^2 + g(\lfloor n/2 \rfloor)$, где $k = \lfloor (n+1)/2 \rfloor$.

Для окончания рекурсии положим $g(0) = 0$.

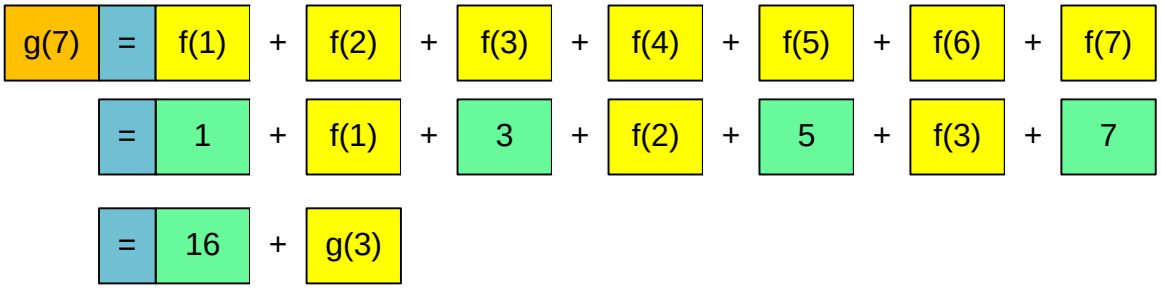
Пример. Рассмотрим первый тест, в котором $n = 7$.

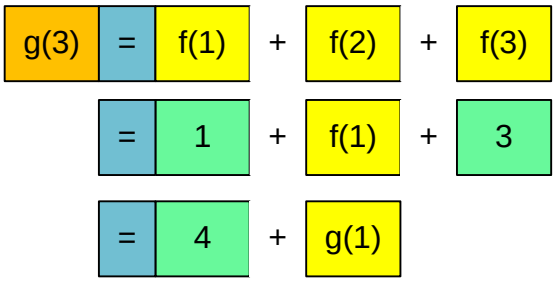
Среди натуральных чисел от 1 до 7 имеется в точности $k = \lfloor (7+1)/2 \rfloor = 4$ нечетных и $l = \lfloor 7/2 \rfloor = 3$ четных числа.



$$g(7) = \left\lfloor \frac{7+1}{2} \right\rfloor^2 + g\left(\left\lfloor \frac{7}{2} \right\rfloor\right) = 16 + g(3) =$$

$$16 + \left\lfloor \frac{3+1}{2} \right\rfloor^2 + g\left(\left\lfloor \frac{3}{2} \right\rfloor\right) = 16 + 4 + g(1) = 16 + 4 + 1 = 21$$





Реализация алгоритма

Реализация функции g приведена ниже.

```
long long g(long long n)
{
  long long k = (n + 1) / 2;
  if (n == 0) return 0;
  return k * k + g(n / 2);
}
```

Основная часть программы. Читаем значение n и выводим $g(n)$.

```
while(scanf("%lld",&n) == 1)
  printf("%lld\n",g(n));
```