# Элементы теории чисел

# Обмен двух переменных

1. `c=a; a=b; b=c;`
2. `a=a+b; b=a-b; a=a-b;`
3. `a=a^b; b=a^b; a=a^b;`
4. `swap(a,b);`

# Теория чисел

Использование теории чисел в олимпиадах по информатике в основном касается:

- общих понятий делимости и деления с остатком;
- простоты чисел, простых множителей;
- наибольшего общего делителя (НОД) и наименьшего общего кратного (НОК);
- моделирования арифметических операций («длинная арифметика»).

# Деление с остатком

```
r = a / b;
q = a % b;
```

Указанные операции не соответствуют математическому определению в случае, когда число a отрицательное (проверьте). Кроме того, они определены и для отрицательных значений b.

# Свойства остатков

- $(a + b) \bmod c = ((a \bmod c) + (b \bmod c)) \bmod c;$
- $(a - b) \bmod c = ((a \bmod c) - (b \bmod c)) \bmod c;$
- $(a * b) \bmod c = ((a \bmod c) * (b \bmod c)) \bmod c.$

# Делители натурального числа

- Делителями натурального числа $a$ называются все натуральные числа, на которые делится $a$ (или, что то же самое, которые делят $a$).
- Для того, чтобы найти все делители числа $a$, можно просматривать на предмет делимости не все числа от 1 до $a$ (следующим улучшением было бы от 1 до a/2 ), а лишь от 1 до $\sqrt{a}$, и для делителей, меньших $\sqrt{a}$, находить пары, большие $\sqrt{a}$

# Поиск делителей числа

```
void divisors(int a)
{
  int i;
  for (i = 1; i * i < a; i++)
    if (a % i == 0)
    {
      process(i); /*обработать делитель*/
      process(a / i);
    }
  if (i * i == a)
    process(i);
}
```

# Простые числа

- Число p называется **простым**, если все его делители - лишь 1 и само число p (ровно два делителя). Если же у числа есть делитель, отличный от 1 и его самого, оно называется **составным**. Число 1 не считается ни простым, ни составным.
- Исходя из этого определения и того, что делители входят в число парами, следует:

  для того, чтобы число p > 1 было простым, достаточно, чтобы у него не было делителей, меньших либо равных $\sqrt{p}$, кроме 1.

# Проверка на простоту

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

# Решето Эратосфена

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |
| 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 |
| 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 |

Prime numbers

# Решето Эратосфена

При реализации алгоритма:

- остановиться можно на числе $\left\lfloor \frac{n}{2} \right\rfloor$ , т.к. далее для любого простого числа первый кандидат на вычеркивание уже больше $n$;
- начинать вычеркивание на шаге, соответствующем простому числу $p$, можно не с $2p$, а с $p^2$, т.к. все меньшие кандидаты на вычеркивание уже были вычеркнуты при обработке меньших простых чисел;
- в силу последнего наблюдения можно вообще остановить работу алгоритма после обработки $p = \lfloor \sqrt{n} \rfloor$ (у каждого составного числа $s \leq n$ есть простой множитель, не больший $\lfloor \sqrt{s} \rfloor \leq \lfloor \sqrt{n} \rfloor$

# Решето Эратосфена

```
void eratosthenes(int n)
{
    /* число 1 не является простым */
    p[1] = false;
    /* сначала все числа "не вычеркнуты" */
    for (int i = 2; i <= n; ++i)
        p[i] = true;
    for (int i = 2; i * i <= n; ++i)
    {
        /* если число простое... */
        if (p[i])
          /*начинаем "вычеркивать" составные числа*/
            for (int j = i * i; j <= n; j += i)
                p[j] = false;
    }
}
```

- Простые числа можно хранить константным массивом в тексте программы:

```
int simple[25]={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97};
```

# Основная теорема арифметики

- Любое натуральное число, большее единицы, может быть однозначно представлено в виде произведения простых сомножителей с точностью до порядка следования;

Иными словами, для любого натурального числа $n > 1$ существует единственный набор упорядоченных по возрастанию различных простых делителей

$$p_1 < p_2 < \ \dots < p_k$$

и набор соответствующих натуральных степеней их вхождения в $n$

$a_1, a_2, \dots, a_k$ , таких что

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ \dots \cdot p_k^{a_k}$$

Например, $12 = 2^2 \cdot 3^1$, $17 = 17^1$, $1\ 073\ 741\ 824 = 2^{30}$.

# Факторизация числа

- Представление числа в виде

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ \dots \cdot p_k^{a_k}$$

называют **факторизацией** числа

# Факторизация числа

```
void factorization(int n)
{
  int a = n; /* копия n, над которой производится деление */
  int p;
  for (p = 2; p * p <= n; ++p)
    if (a % p == 0)
    {
       int c = 0; /* счетчик степени вхождения числа */
       while (a % p == 0)
       {
          a /= p;
          c++;
       }
       report(p, c);
    }
  if (a != 1)
    report(a, 1);
}
```

- Из основной теоремы арифметики и определения делимости следует, что одно число делится на другое тогда и только тогда, когда набор простых множителей первого (с учетом кратностей $i$) целиком содержит в себе набор простых множителей последнего.

Это значит, что при

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ \dots \cdot p_k^{a_k}$$

все его делители имеют вид

$$m = p_1^{b} \cdot p_2^{b_2} \cdot \ \dots \cdot p_k^{b_k}, \ 0 \le b_i \le a_i, \ i = \overline{1,k}$$

и определяются однозначно набором $b_i$.

- Отсюда, в частности, следует, что количество различных делителей числа $n$ при принятых обозначениях равно количеству всевозможных комбинаций $b_i$.

Каждое $b_i$ может независимо принимать $a_i + 1$ значений (от 0 до $a_i$), а потому общее число делителей есть

$$\sigma(n) = (a_1 + 1)(a_2 + 1) \ldots (a_k + 1)$$

# Хранение числа в виде разложения

- Любое число можно представить в виде произведения простых чисел. Такое представление выглядит в виде одномерного массива в каждой ячейке которого содержится степень соответствующего просто числа.
- Например, пусть у нас есть массив простых чисел

[2; 3; 5; 7; 11; 13],

  - тогда число 2600 будет выглядеть как [3; 0; 2; 0; 0; 1]

  $(2^3 \cdot 5^2 \cdot 13^1 = 2600)$,

  - число 11858 представляется массивом [1; 0; 0; 2; 2; 0]

  $(2^1 \cdot 7^2 \cdot 11^2 = 11858)$

# Хранение числа в виде разложения

Такие числа легко умножать. Чтобы получить произведение чисел, достаточно сложить соответствующие элементы массивов.

[3; 0; 2; 0; 0; 1]

[1; 0; 0; 2; 2; 0]

Результатом будет массив [4; 0; 2; 2; 2; 1], т.е. $2^4 \cdot 5^2 \cdot 7^2 \cdot 11^2 \cdot 13^1 = 30830800 = 2600 \cdot 11858$.

Можно реализовать деление с остатком, нахождение НОД и НОК.

# НОД и НОК

- **Наибольшим общим делителем (НОД)** неотрицательных целых чисел $a$ и $b$ (не являющихся одновременно нулями) назовем наибольший элемент множества общих делителей чисел $a$ и $b$.
- Общим кратным двух натуральных чисел $a$ и $b$ называется любое такое натуральное число с, что а делит $c$ и $b$ делит $c$. **Наименьшим общим кратным (НОК)** называется минимальное из таких натуральных чисел

# НОД и НОК

- Пусть числа заданы разложением на простые множители.
- Разложение на простые множители НОД(a; b) включает в себя только те простые числа, которые встречаются в как в разложении a, так и в разложении b. При этом степень, в которой простое число входит в разложение НОД, равна минимальной из степеней, в которых число входит в разложения a и b.
- НОК двух чисел состоит из тех простых множителей, которые встречаются хотя бы в одном из разложений исходных чисел, а кратность (степень) вхождения такого множителя равна максимальной из его кратностей в разложениях исходных чисел.

# НОД и НОК

- Пусть есть два числа $n$ и $m$

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ \dots \cdot p_k^{a_k}$$
$$m = p_1^{b} \cdot p_2^{b_2} \cdot \ \dots \cdot p_k^{b_k}$$

- Тогда

$$\text{НОД}\ (n,\ m) = p_1^{\min(a_1,\ b_1)} \cdot p_2^{\min(a_2,\ b_2)} \cdot \ \dots \cdot p_k^{\min(a_k,\ b_k)}$$
$$\text{НОК}\ (n,\ m) = p_1^{\max(a_1,\ b_1)} \cdot p_2^{\max(a_2,\ b_2)} \cdot \ \dots \cdot p_k^{\max(a_k,\ b_k)}$$

# Свойства НОД

- НОД(a; a) = a;
- НОД(a; 1) = 1;
- НОД(a; 0) = a:
- НОД(a, b) = НОД(a – b, b)
- НОД(a, b) = НОД(a mod b, b)
- НОД(2a, 2b) = 2НОД(a, b)
- НОД(2a, b) = НОД(a, b), если b – нечетно.

# Алгоритм Евклида

```
int gcd (int a, int b)
{
    while (b && a)
    {
        if (a > b)
          a %= b;
        else
          b %= a;
    }
    return a + b;
}
```

# Варианты реализации алгоритма Евклида

1. нахождение разностей
2. обмен значений
3. рекурсивная реализация
4. рекурсия с тринарным оператором
5. бинарная реализация
6. использование библиотеки

# Евклид (нахождение разностей)

```
int gcd (int a, int b)
{
    while (b!=a)
    {
        if (a>b)
         a -= b;
        else
         b -= a;
    }
    return a;
}
```

# Евклид (обмен значений)

```
int gcd (int a, int b)
{
    while (b)
    {
        a %= b;
        swap (a, b);
    }
    return a;
}
```

# Евклид (рекурсивная реализация)

```
int gcd (int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd (b, a % b);
}
```

# Евклид (рекурсия с тринарным оператором)

```
int gcd (int a, int b)
{
    return b ? gcd (b, a % b) : a;
}
```

# Евклид (бинарная реализация)

```
int gcd (int a, int b)
{
    int c = 1;
    while (b && a)
    {
        unsigned int d = b & 1;
        if (!(a & 1))
          if (!d)
          {
              a >>= 1; b >>= 1; c <<= 1;
          }
          else a >>= 1;
        else
           if (!d) b >>= 1;
           else
              if (a > b)
                a %= b;
              else
                b %= a;
    }
    return (a + b) * c;
}
```

# Евклид (использование библиотеки)

```
#include <iostream>
#include <algorithm>
#include <numeric>

using namespace std;

int main()
{
    cout << __gcd(78,56) << endl;
    return 0;
}
```

# Функция Эйлера

- **Функция Эйлера** $\varphi(n)$ - это количество чисел от 1 до $n$, взаимно простых с $n$.
- Иными словами, это количество таких чисел в отрезке [1; $n$], наибольший общий делитель которых с $n$ равен единице.
  - $\varphi(1)=1$
  - $\varphi(2)=1$
  - $\varphi(3)=2$
  - $\varphi(4)=2$
  - $\varphi(5)=4$

# Свойства функции Эйлера

- Если p – простое число, то $\varphi(p) = p - 1$
  - Это очевидно, т.к. любое число, кроме самого $p$, взаимно просто с ним.
- Если p – простое число, а $a$ – натуральное число, то
$$\varphi(p^a) = p^a - p^{a-1}$$
  - Поскольку с числом $p^a$ не взаимно просты только числа вида $(pk, k \in \mathcal{N})$, которых $\frac{p^a}{p} = p^{a-1}$ штук.
- Если $a$ и $b$ взаимно простые, то $\varphi(ab) = \varphi(a)\varphi(b)$ ("мультипликативность" функции Эйлера).

# Вычисление функции Эйлера

- Пусть $n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ \dots \cdot p_k^{a_k}$
- Тогда

$$\varphi(n) = \varphi\left(p_1^{a_1}\right) \cdot \varphi\left(p_2^{a_2}\right) \cdot \ \dots \cdot \varphi\left(p_k^{a_k}\right) =$$

$$= (p_1^{a_1} - p_1^{a_1-1}) \cdot (p_2^{a_2} - p_2^{a_2-1}) \cdot \ \dots \cdot \left(p_k^{a_k} - p_k^{a_k-1}\right) =$$

$$= n(1 - \frac{1}{p_1})(1 - \frac{1}{p_1}) \cdot \ \dots \cdot (1 - \frac{1}{p_k}) =$$

$$= \frac{n}{p_1 \cdot p_2 \cdot \dots \cdot p_k} (p_1 - 1)(p_2 - 1) \cdot \ \dots \cdot (p_k - 1)$$

# Вычисление функции Эйлера

```
int phi (int n)
{
    int result = n;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
        {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    if (n > 1)
        result -= result / n;
    return result;
}
```

# Ввод – Вывод на C++

```
#include <cstdio>

using namespace std;

int main()
{
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int a,b;
    scanf("%d%d", &a, &b);
    printf("%d", a + b);
    return 0;
}
```

```
#include <fstream>

using namespace std;

int main()
{
   ifstream fin("input.txt");
   ofstream fout("output.txt");
    int a, b;
   fin >> a >> b;
   fout << a + b;
   return 0;
}
```

| Код | действие |
| --- | --- |
| %c | Читает один символ |
| %d | Читает десятичное целое |
| %f | Читает вещественное число |
| %s | Читает строку |
| %u | Читает десятичное целое без знака |
| %lld | Читает целое типа long long () – Linux |
| %I64d | Читает целое типа long long () - Windows |

Универсальный" код, который работает правильно под обеими системами, может выглядеть так:

```
#ifdef WIN32
    printf("%I64d\n",ans);
#else
    printf("%lld\n",ans);
#endif
```

```
ios_base::sync_with_stdio(0);
```

- Для ускорения ввода-вывода при использовании потокового ввода-вывода
- Не использовать вместе с:
  - freopen
  - #include <cstdio>

# ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ

- #include <bits/stdc++.h>