Основы программирования на Java

1. Алгоритм Евклида
2. Решето Эратосфена
3. Длинные числа

Целочисленные алгоритмы

Тема 1. Алгоритм Евклида

Вычисление НОД

НОД = наибольший общий делитель двух натуральных чисел – это наибольшее число, на которое оба исходных числа делятся без остатка.

Перебор:

```
k = a;  // или k = b;
while ( a % k != 0  ||
        b % k != 0 )
  k --;
printf ("НОД(%d,%d)=%d", a, b,
k);
```

или

много операций для больших чисел

Алгоритм Евклида

```
НОД(a,b)=НОД(a-b,b)
        =НОД(a,b-a)
```

Евклид
(365-300 до. н. э.)

Заменяем большее из двух чисел разностью большего и меньшего до тех пор, пока они не станут равны. Это и есть НОД.

НОД вычисляется через НОД. Как это называется?

Пример:

```
НОД (14, 21) = НОД (14, 21-14) = НОД (14,
7)           = НОД (7, 7) = 7
```

много шагов при большой разнице чисел:

```
НОД (1998, 2) = НОД (1996, 2) = … =
2
```

Модифицированный алгоритм Евклида

```
НОД(a,b)=НОД(a%b,b)
        =НОД(a, b%a)
```

Заменяем большее из двух чисел остатком от деления большего на меньшее до тех пор, пока меньшее не станет равно нулю. Тогда большее — это НОД.

Пример:

```
НОД (14, 21) = НОД (14, 7) = НОД (0, 7) =
7
```

Еще один вариант:

```
НОД(2·a,2·b)= 2·НОД(a,b)
НОД(2·a,b)= НОД(a,b)  // при нечетном b
```

Реализация алгоритма Евклида

Рекурсивный вариант:

```
static int gcd(int a,int b)
{
 if ( a == b ) return a;
 if ( a < b )
      return gcd ( a, b-a );
 else return gcd ( a-b, b );
}
```

Без рекурсии:

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

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

```
static int gcd(int a, int b)
{
  while ( a*b != 0 )
    if ( a > b ) a = a % b;
    else         b = b % a;
  return a + b;
}
```

Почему `return a+b`?

Задания

«4»: Составить программу для вычисления НОД и заполнить таблицу:

N	64168	358853	6365133	17905514	549868978
M	82678	691042	11494962	23108855	298294835
НОД(N,M)					

«5»: То же самое, но сравнить для каждой пары число шагов обычного и модифицированного алгоритмов (добавить в таблицу еще две строчки).

Целочисленные алгоритмы

Тема 2. Решето Эратосфена

Поиск простых чисел

Простые числа – это числа, которые делятся только на себя и на 1.

Применение:

1) криптография;
2) генераторы псевдослучайных чисел.

Наибольшее известное (26 декабря 2017):

$2^{77\,232\,917} - 1$ (содержит 23 249 425 цифр).

Задача. Найти все простые числа в интервале от 1 до заданного N.

Простое решение:

```
for ( i = 1; i <= N; i++ ) {
  isPrime = true;
  for ( k = 2; k*k <= i; k++ )
    if ( i % k == 0 ) {
      isPrime = false;
      break;
      }
  if ( isPrime )
    printf("%d\n", i);
  }
```

Как уменьшить число шагов внутреннего цикла?

$k <= \sqrt{i}$ ⟺ `k*k <= i`

Как оценить число операций?

$O(N^2)$ растет не быстрее N^2

Решето Эратосфена

Эратосфен Киренский
(Eratosthenes, Ερατοσθδνη)
(ок. 275-194 до н.э.)

Алгоритм:

1) начать с **k = 2**;
2) «выколоть» все числа через **k**, начиная с **2·k**;
3) перейти к следующему «невыколотому» **k**;
4) если **k·k <= N**, то перейти к шагу 2;
5) напечатать все числа, оставшиеся «невыколотыми».

Новая версия – решето Аткина .

высокая скорость, количество операций

$$O((N \cdot \log N) \cdot \log \log N)$$

нужно хранить в памяти все числа от **1** до **N**

Реализация

Массив **A[N+1]**, где

A[i]=true, если число **i** не «выколото»,

A[i]=false, если число **i** «выколото».

```
// сначала все числа не выколоты
for ( i = 1; i <= N; i ++ )
  A[i] = true;
// основной цикл
for ( k = 2; k*k <= N; k ++ )
  if ( A[k] )
    for ( i = k+k; i <= N; i += k ) A[i] = false;
// выводим оставшиеся числа
for ( i = 1; i <= N; i ++ )
  if ( A[i] )
    printf ( "%d\n", i );
```

Задания

«4»: Реализовать «решето Эратосфена», число *N* вводить с клавиатуры.

«5»: То же самое, но сравнить число шагов алгоритма для различных значений *N*. Построить график в *Excel*, сравнить сложность с линейной. Заполнить таблицу:

N	1000	5000	10000	20000	50000
Количество простых чисел					
Число шагов внутреннего цикла					

Целочисленные алгоритмы

Тема 3. Длинные числа

Что такое длинные числа?

Задача. Вычислить (точно)

$$100! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot 99 \cdot 100$$

Проблема:

это число содержит более 100 цифр…

Сколько нулей в конце этого числа?

Какая последняя ненулевая цифра?

Решение:

хранить цифры в виде массива, по группам (например, 6 цифр в ячейке).

Сколько ячеек нужно?

$100! < \underbrace{100^{100}}_{\text{201 цифра}}$

$201/6 \approx 34$ ячейки

Хранение длинных чисел

$$1234\ 568901\ 734567 =$$
$$= 1234 \cdot 1000000^2 + 568901 \cdot 1000000^1 + 734567 \cdot 1000000^0$$

На что это похоже?

Хранить число по группам из 6 цифр – это значит представить его в системе счисления с основанием **d** = 1000000.

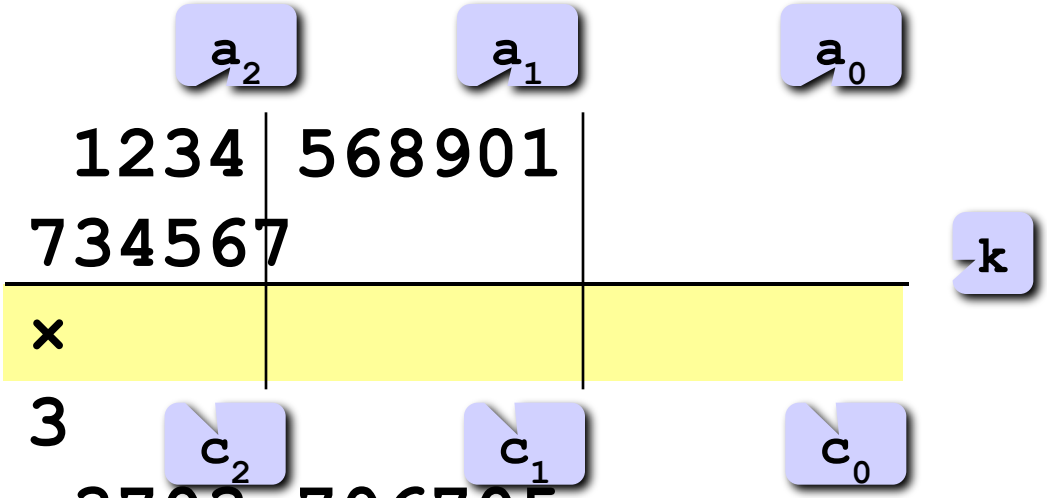
Алгоритм:

{A} – длинное число, хранящееся как массив

```
{A} = 1;
for ( k = 2; k <= 100; k ++ )
   { A} = {A} * k;
... // вывести { A}
```

умножение длинного числа на «короткое»

Умножение длинного числа на короткое



$734567 \cdot 3 = 2203701$

$20\ldots$

перенос, r_1

c_0

$568901 \cdot 3 + 2 = 1706705$

r_2 c_1

$1234 \cdot 3 + 1 = 3703$

c_2

$c_0 = (a_0 \cdot k + 0) \% d$
$r_1 = (a_0 \cdot k + 0) / d$

$c_1 = (a_1 \cdot k + r_1) \% d$
$r_2 = (a_1 \cdot k + r_1) / d$

$c_2 = (a_2 \cdot k + r_2) \% d$
$r_3 = (a_2 \cdot k + r_2) / d$
...

Вычисление 100!

```
int d = 1000000;          // основание системы
int s, r;                 // произведение, остаток
int[] A = new int[40];
A[0] = 1;                 // A[0]=1, остальные A[i]=0
int i, k, len = 1;        // len – длина числа
for ( k = 2; k <= 100; k ++ ) {
  i = 0;
  r = 0;
  while ( i < len  ||  r > 0 ) {
    s = A[i]*k + r;
    A[i] = s % d;      // остается в этом разряде
    r = s / d;         // перенос
    i ++;
    }
  len = i;      // новая длина числа
  }
```

пока не кончились цифры числа **A** или есть перенос

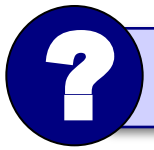
Где результат?

Можно ли брать другое d?

Как вывести длинное число?

«Первая мысль»:

```
for ( i = len-1; i >= 0; i -- )
  printf ( "%d", A[i] );
```



Что плохо?

Проблема:

как не потерять первые нули при выводе чисел, длина которых менее 6 знаков?

123 → 000123

Решение:

1) составить свой метод;
2) использовать формат **"%06d"**!

```
for ( i = len-1; i >= 0; i -- )
  if ( i == len-1 ) printf ( "%d", A[i] );
  else              printf ( "%06d", A[i] );
```