

# Целочисленные алгоритмы (язык Паскаль)

1. Алгоритм Евклида
2. Решето Эратосфена
3. Длинные числа
4. Целочисленная оптимизация

# Целочисленные алгоритмы (язык Паскаль)

## Тема 1. Алгоритм Евклида

# Вычисление НОД

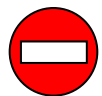
---

**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 mod k <> 0) or  
      (b mod k <> 0) do  
  k := k - 1;  
writeln ('НОД(' , a , ',' , b , ')=' , k);
```

или



много операций для больших чисел

# Алгоритм Евклида

$$\begin{aligned}\text{НОД}(a, b) &= \text{НОД}(a-b, b) \\ &= \text{НОД}(a, b-a)\end{aligned}$$

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



Евклид  
(365-300 до. н. э.)

**?** НОД вычисляется через НОД. Как это называется?

**Пример:**

$$\begin{aligned}\text{НОД}(14, 21) &= \text{НОД}(14, 21-14) = \text{НОД}(14, \\ 7) &= \text{НОД}(7, 7) = 7\end{aligned}$$

**⊖** много шагов при большой разнице чисел:

$$\begin{aligned}\text{НОД}(1998, 2) &= \text{НОД}(1996, 2) = \dots = \\ 2\end{aligned}$$

# Модифицированный алгоритм Евклида

---

$$\begin{aligned}\text{НОД}(a, b) &= \text{НОД}(a \bmod b, b) \\ &= \text{НОД}(a, b \bmod a)\end{aligned}$$

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:**

$$\text{НОД}(14, 21) = \text{НОД}(14, 7) = \text{НОД}(0, 7) =$$

**Еще <sup>7</sup> один вариант:**

$$\text{НОД}(2 \cdot a, 2 \cdot b) = 2 \cdot \text{НОД}(a, b)$$

$$\text{НОД}(2 \cdot a, b) = \text{НОД}(a, b) \quad // \text{ при нечетном } b$$

# Реализация алгоритма Евклида

## Рекурсивный вариант:

```
function NOD (a, b: integer): integer;  
begin  
  if a = b then NOD := a  
  else  
    if a < b then  
      NOD := NOD(a, b-a)  
    else NOD := NOD(a-b, b);  
end;
```

```
function NOD (a, b: integer): integer;  
begin  
  if a*b = 0 then NOD := a+b  
  else  
    if a < b then  
      NOD := NOD(a, b mod a)  
    else NOD := NOD(a mod b, b);  
end;
```



Почему a+b?

# Реализация алгоритма Евклида

---

## Без рекурсии:

```
function NOD (a, b: integer): integer;  
begin  
  while a <> b do  
    if a > b then a := a - b  
    else          b := b - a;  
  NOD := a;  
end;
```

```
function NOD (a, b: integer): integer;  
begin  
  while a*b <> 0 do  
    if a > b then a := a mod b  
    else          b := b mod a;  
  NOD := a + b;  
end;
```

# 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 всех пар число шагов обычного и модифицированного алгоритмов (добавить в таблицу еще две строчки).



# Целочисленные алгоритмы (язык Паскаль)

## 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 (сентябрь 2008):**

$2^{43112609} - 1$  (содержит 12 978 189 цифр).

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

**Простое решение:**

```
for i:=1 to N do begin
  isPrime := True;
  for k:=2 to i-1 do
    if i mod k = 0 then
      isPrime := False;
  if isPrime then
    writeln(i);
end;
```



Как уменьшить число шагов внутреннего цикла?

$$k \leq \sqrt{i} \quad \longleftrightarrow \quad k * k \leq i$$



Как оценить число операций?

$O(N^2)$  растёт не быстрее  $N^2$

# Решето Эратосфена



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

## Алгоритм:

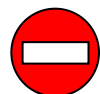
- 1) начать с  $k = 2$ ;
- 2) «выколоть» все числа через  $k$ , начиная с  $2 \cdot k$ ;
- 3) перейти к следующему «невыколотому»  $k$ ;
- 4) если  $k \cdot k \leq N$ , то перейти к шагу 2;
- 5) напечатать все числа, оставшиеся «невыколотыми».

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

---

Логический массив  $A[N]$ , где

$A[i] = \text{True}$ , если число  $i$  не «выколото»,

$A[i] = \text{False}$ , если число  $i$  «выколото».

```
{ сначала все числа не выколоты }  
for i:=1 to N do A[i] := True;  
  
{ основной цикл  
  «выкалывание» составных чисел }  
  
{ выводим оставшиеся числа }  
for i:=1 to N do  
  if A[i] then writeln(i);
```

# Реализация

## Основной цикл:

```
k := 2;
while k*k <= N do begin
  if A[k] then begin
    i := k*k;
    while i <= N do begin
      A[i] := False;
      i := i + k;
    end;
  end;
  k := k + 1;
end;
```

«ВЫКАЛЫВАЕМ»  
все числа,  
кратные k

# 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:

<b>N</b>	1000	5000	10000	20000	50000
<b>Количество простых чисел</b>					
<b>Число шагов внутреннего цикла</b>					

# Целочисленные алгоритмы (язык Паскаль)

## Тема 3. Длинные числа

# Что такое длинные числа?

**Задача.** Вычислить (точно)

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

**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?**

$$201 / 6 \approx 34 \text{ ячейки}$$

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



# Хранение длинных чисел

$$\begin{aligned}
 &1234 \ 568901 \ 734567 = \\
 &= 1234 \cdot 1000000^2 + \\
 &568901 \cdot 1000000^1 + \\
 &734567 \cdot 1000000^0
 \end{aligned}$$



На что это похоже?

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

## Алгоритм:

```

[A] = 1;
for k := 2 to 100 do
  [A] = [A] * k;
{ вывести [A] }

```

[A] – длинное число, хранящееся как массив

умножение длинного числа на «короткое»

# Умножение длинного числа на короткое

$a_2$        $a_1$        $a_0$   
 1234 | 568901  
 734567  
 —————  $k$   
 x  
 3       $c_2$        $c_1$        $c_0$   
 3703 | 706705  
 734567 · 3 = 2023701  
 2023701  
 перенос,  $r_1$        $c_0$   
 568901 · 3 + 2 = 1  
 706705  
 $r_2$        $c_1$   
 1234 · 3 + 1 =  
 $c_2$   
 3703

$$c_0 = (a_0 \cdot k + 0) \bmod d$$

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

$$c_1 = (a_1 \cdot k + r_1) \bmod d$$

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

$$c_2 = (a_2 \cdot k + r_2) \bmod d$$

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

...

# Вычисление 100!

```
const d = 1000000; { основание системы }
var A: array[0..40] of integer;
    s, r,           { произведение, остаток }
    i, k,           { вспомогательные }
    len: integer;  { длина числа }
begin
    { присвоить [A] = 1 }
    { последовательно умножать
      [A] на 2, 3, ..., 100 }
    { вывести [A] }
end.
```

# Вычисление 100!

```

len := 1;           { записать [A]=1 }
A[0] := 1;
for i:=1 to 40 do
  A[i] := 0;

for k:=2 to 100 do begin
  i := 0; { с младшего разряда }
  r := 0; { пока нет переноса }
  while (i < len) or (r > 0) do begin
    s := A[i]*k + r;
    A[i] := s mod d; { в этом разряде }
    r := s div d;    { перенос }
    i := i + 1; { к следующему разряду }
  end;
  len := i;        { новая длина числа }
end;
end;

```

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 downto 0 do  
  write(A[i]);
```



Что плохо?

## Проблема:

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

123 → 000123

## Решение:

составить свою процедуру, а при выводе старшего разряда ( $len-1$ ) убирать лидирующие нули:

```
write(A[len-1]); { старший разряд }  
for i:=len-2 downto 0 do  
  Write6(A[i]);
```



# Задания

---

**«4»:** Составить программу для вычисления

$$99!! = 1 \cdot 3 \cdot \dots \cdot 97 \cdot 99$$

**«5»:** То же самое, но написать свою процедуру для вывода, использующую символьные строки.

**«6»:** Написать программу для умножения двух длинных чисел (ввод из файла).

**«7»:** Написать программу для извлечения квадратного корня из длинного числа (ввод из файла).

# Целочисленные алгоритмы (язык Паскаль)

## Тема 4. Целочисленная оптимизация



# Задачи целочисленной оптимизации

---

## Оптимизация:

$f(x) \rightarrow \min$  при заданных ограничениях

## Целочисленная оптимизация:

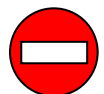
$x$  – вектор (массив) целых чисел

## Комбинаторная оптимизация:

$x$  – вектор (массив) целых чисел, причем все его элементы принадлежат заданному набору чисел



при малом количестве вариантов можно решить простым перебором



при большом количестве вариантов на решение перебором может потребоваться огромное время (для ряда задач другие алгоритмы неизвестны)

# Задача коммивояжера

**Задача коммивояжера.** Коммивояжер (бродячий торговец) должен выйти из первого города и, посетив по разу в неизвестном порядке города  $2, 3, \dots, N$ , вернуться обратно в первый город. В каком порядке надо обходить города, чтобы замкнутый путь (тур) коммивояжера был кратчайшим?



**Это NP-полная задача, которая строго решается только перебором вариантов (пока)!**

## Точные методы:

- 1) простой перебор;
- 2) метод ветвей и границ;
- 3) метод Литтла;
- 4) ...



большое время счета для больших  $N$

$O(N!)$

## Приближенные методы:

- 5) метод случайных перестановок (*Matlab*);
- 6) генетические алгоритмы;
- 7) метод муравьиных колоний;
- 8) ...



не гарантируется оптимальное решение

# Метод случайных перестановок

Что представляет собой решение?

перестановка чисел  $2, 3, \dots, N$ .



комбинаторная  
задача

Алгоритм:

- 1) записать в массив  $x$  перестановку

2 3 ... N

найти длину маршрута

1 → 2 → 3 → ... → N → 1

и записать ее в  $L_{min}$ ;

- 2) выбрать случайно два элемента массива  $x$  и поменять их местами;
- 3) найти длину маршрута, соответствующего  $x$  и, если она меньше  $L_{min}$ , записать ее в  $L_{min}$  и запомнить перестановку;
- 4) если число шагов меньше заданного, перейти к шагу 2.

# Конец фильма

---