Целочисленные алгоритмы (язык Си)

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

Целочисленные алгоритмы (язык Си)

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:

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

Без рекурсии:

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

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

```
int NOD ( 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 (язык Си)

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

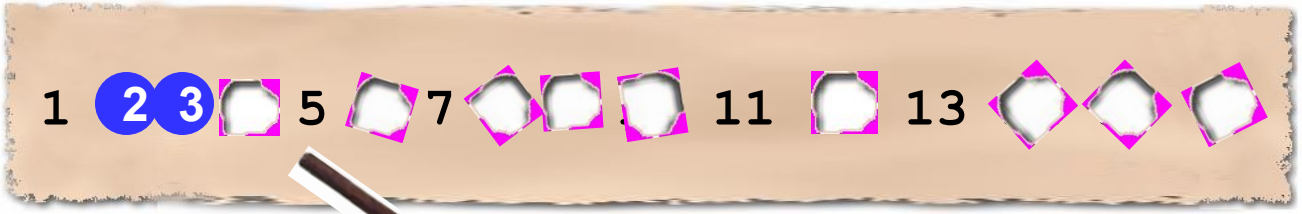
Как уменьшить число шагов внутреннего цикла?

$k <= \sqrt{i}$ ⟺ `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]=1, если число **i** не «выколото»,

A[i]=0, если число **i** «выколото».

```
// сначала все числа не выколоты
for ( i = 1; i <= N; i ++ )
  A[i] = 1;
// основной цикл
for ( k = 2; k*k <= N; k ++ )
  if ( A[k] != 0 )
    for ( i = k+k; i <= N; i += k ) A[i] = 0;
// выводим оставшиеся числа
for ( i = 1; i <= N; i ++ )
  if ( A[i] == 1 )
    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 (язык Си)

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.

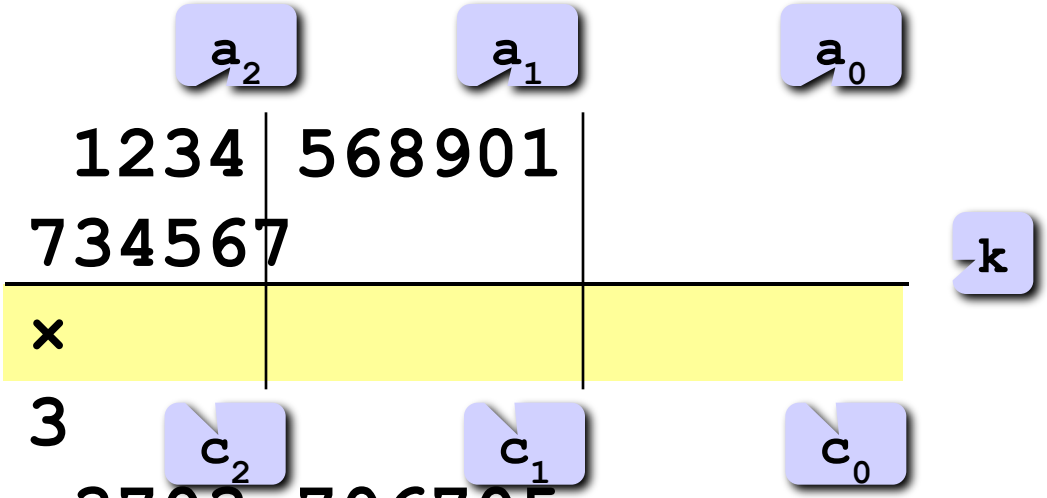
Алгоритм:

{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!

```
const int d = 1000000;    // основание системы
int  A[40] = {1},         // A[0]=1, остальные A[i]=0
     s, r;                // произведение, остаток
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?

Вариант для C++

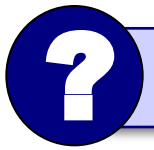
```
#include <vector>
typedef vector<int> lnum;
const int d = 1000000;    // основание системы
lnum  A = {1};
int s, r;                 // произведение, остаток
int i, k, len = 1;        // len – длина числа

for ( k = 2; k <= 100; k ++ ) {
  i = 0;
  r = 0;
  while ( i < A.size()  ||  r > 0 ) {
    if (i==A.size()) a.pushback(0);
    s = A[i]*k + r;
    A[i] = s % d;       // остается в этом разряде
    r = s / d;          // перенос
    i ++;
    }
  while (A.size() > 1 && A.back() == 0) A.pop_back();
  }
```

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 свою процедуру;
2) использовать формат **"%.6d"**!

```
for ( i = len-1; i >= 0; i -- )
  if ( i == len-1 ) printf ( "%ld", A[i] );
  else              printf ( "%.6d", A[i] );
```

Вариант для C++

Сначала мы просто выводим самый последний элемент вектора (или , если вектор пустой), а затем выводим все оставшиеся элементы вектора, дополняя их нулями до 6 символов:

```
printf ("%d", A.empty() ? 0 : a.back());
for (int i=(int)A.size()-2; i>=0; --i)
printf ("%06d", a[i]);
```

здесь небольшой тонкий момент: нужно не забыть записать приведение типа (int), поскольку в противном случае число будут беззнаковым, и если , то при вычитании произойдёт переполнение

Задания

«4»: Составить программу для вычисления

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

«5»: То же самое, но написать свою процедуру для вывода (не использовать формат "%.6d").

“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 Си)

Тема 4. Целочисленная оптимизация

Задачи целочисленной оптимизации

Оптимизация:

$$f(x) \to \min$$ 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)

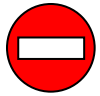
Задача коммивояжера

Задача коммивояжера. Коммивояжер (бродячий торговец) должен выйти из первого города и, посетив по разу в неизвестном порядке города **2,3,...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?

комбинаторная задача

перестановка чисел **2,3,...N.**

Алгоритм:

1) записать в массив x перестановку

   2 3 … N

   найти длину маршрута

   1 ➡ 2 ➡ 3 ➡ … ➡ N ➡ 1

   и записать ее в **Lmin**;

2) выбрать случайно два элемента массива x и поменять их местами;

3) найти длину маршрута, соответствующего x и, если она меньше **Lmin**, записать ее в **Lmin** и запомнить перестановку;

4) если число шагов меньше заданного, перейти к шагу 2.

Конец фильма