УДК 004.021

АЛГОРИТМЫ НАХОЖДЕНИЯ ПОСЛЕДНЕЙ НЕНУЛЕВОЙ ЦИФРЫ ФАКТОРИАЛА ЧИСЛА ПОРЯДКА 10^9

Атаев Б.К., Мухарский Д.В., Карымсаков Ж.Ж.

КГУ «Кокшетауский государственный университет им. Ш. Уалиханова», Кокшетау, e-mail: mail@kgu.kz, kgu@mail.kz

Рассматриваются три алгоритма для нахождения последней ненулевой цифры факториала от больших чисел порядка 10^9. Алгоритмы рассмотрены в порядке увеличения их быстродействия. Рассмотрены элементы реализации алгоритмов на языке программирования Turbo Pascal. В конце статьи приводятся сравнительные характеристики времени выполнения каждого из рассмотренных алгоритмов.

Ключевые слова: факториал, последняя ненулевая цифра

ALGORITHMS FIND THE LAST ON-ZERO DIGIT FACTORIAL OF THE DECIMAL EXPONENT OF 10^9

Atayev B.K., Muharsky D.V., Karymsakov J.J.

Sh. Ualikhanov Kokshetau State University, Kokshetau, e-mail: mail@kgu.kz, kgu@mail.kz

Considered three algorithms for finding the last non-zero digit of the factorial of large numbers of the decimal exponent of 10^9. Algorithms are considered in order of increasing performance. Examine the elements of the implementation of algorithms in a programming language Turbo Pascal. The article shows the comparative performance of time for each of the considered algorithms.

Keywords: factorial, the last non-zero digit

Найти алгоритм, определяющий младшую ненулевую цифру в десятичной записи числа $N!=1\cdot2\cdot3\cdot\ldots\cdot N$ $(1\leq N\leq 2\cdot10^9)$.

Для каждой задачи всегда можно найти несколько решений, одни из них очевидные, другие нет. Есть задачи, трудность решения которых растёт в экспоненциальной зависимости от исходных данных. Проблема вычисления факториала большого числа относится именно к этому типу. Но нахождение последней ненулевой цифры факториалов даже очень больших чисел легко осуществляется на любом компьютере. Всё дело в грамотном подборе алгоритма. Разобранные в статье алгоритмы демонстрируют различные подходы к решению, от решения, что называется «в лоб», до тонкого наблюдения над свойствами чисел, позволяющего решить задачу за доли секунды. Думается, статья будет полезна для всех интересующихся нестандартными алгоритмами.

Первый алгоритм

Данный алгоритм можно считать наиболее точным. Он основывается на принципах так называемой длинной арифметики. Никакой целочисленный тип не способен вместит в себя факториалы даже небольших чисел, не говоря уже о факториалах чисел порядка 10^9. Единственным выходом здесь может быть использование для хранения столь длинных чисел массивов. К массивам можно применить правила сложения и перемножения чисел школьным способом – в столбик.

При вычислении факториала некоторого числа длина результата будет постоянно увеличиваться. Таким образом, необходимо либо заранее зарезервировать очень длинный массив, либо использовать динамические массивы, что сильно замедлит работу. Но в Delphi есть прекрасное хранилище для чисел, какого угодно размера – тип string. Его размер 2 Гигабайта, что достаточно для хранения числа длиной 1 073 741 824 знака.

Алгоритм имитирует перемножение чисел в столбик. Если необходимо перемножить два числа, представленных символьными массивами, поступаем следующим образом:

- Перемножаем самую правую цифру второго числа с первым числом.
- Перемножаем вторую с конца цифру второго числа с первым числом.
- Складываем два получившихся числа с учётом сдвига на один разряд.
- Продолжаем операцию до тех пор, пока не закончатся цифры во втором числе.
- Пример реализации данного алгоритма приведён ниже.

						7	4	8	2	4	
×					2	3	0	8	3	4	
					2	9	9	2	9	6	
+				2	2	4	4	7	2		
				2	5	4	4	0	1	6	
+			5	9	8	5	9	2			
			6	2	4	0	3	2	1	6	
+			0	0	0	0	0				
			6	2	4	0	3	2	1	6	
+		2	2	4	4	7	2				
		2	3	0	7	1	2	3	2	1	6
+	1	4	9	6	4	8					
	1	7	2	7	1	9	2	3	2	1	6

Рис. 1. Пример выполнения умножения «столбиком»

Практическая реализация алгоритма состоит из трёх вспомогательных функций. Первая функция предназначена для умножения цифры на число любой длины. Для лучшего понимания функция приводится с незначащими сокращениями:

```
// УМНОЖЕНИЕ ЦИФРЫ НА МНОГОЗНАЧНОЕ ЧИСЛО
// ВХОД:  цифра
//        число (обратный порядок)
// ВЫХОД: строка результата (обратный порядок)
function multy(sNum1, sNum2: string): string;
var
  k: integer;
  inMind: integer;    // то, что в уме
  inFigure: integer;  // произведение
  sFigure: string[2];// произведение в символьном виде
begin
  inMind := 0;
  result := '';
  for k := 1 to length(sNum2) do
  begin
    inFigure := StrToInt(sNum1)*StrToInt(sNum2[k]);
    inFigure := inFigure + inMind;
    sFigure := IntToStr(inFigure);
    if length(sFigure) = 1 then
    begin
      result := result + sFigure[1];
      inMind := 0;
    end
    else
    begin
      result := result + sFigure[2];
      inMind := StrToInt(sFigure[1]);
    end;
  end;
  if inMind <> 0 then
    result := result + IntToStr(inMind);
end;
```

Она принимает в качестве аргументов цифру и строку, содержащую число и возвращает строку, содержащую произведение. При перемножении чисел все действия производятся с конца. При программировании перебирать символы строки легче, начиная с первого. Поэтому передавать аргументы в функцию и возвращать результат легче в обратном виде. Приведение к нормальной форме происходит только при выводе результата.

Алгоритм работы функции следующий:

- Запускаем цикл по длине второй строки.
- Перемножаем цифру и очередную цифру числа.
- Прибавляем к результату то, что запомнили от предыдущего действия.
- Если результат однозначный, приписываем его к результату, а если двузначный, то первую цифру приписываем к результату, а вторую запоминаем для последующего действия.
- Берём следующую цифру.
- После завершения цикла, если «в уме» осталась цифра, приписываем её к результату.

Следующей вспомогательной функцией является сложение двух чисел произвольной длины. Ввиду громоздкости функции не представляется возможным привести её полностью. Ниже представлены только заголовок функции и алгоритм её работы:

```
// СЛОЖЕНИЕ ДВУХ ЧИСЕЛ
// ВХОД:  два слагаемых  (обратный порядок)
//        смещение разряда
// ВЫХОД: строка результата (обратный порядок)
function Add(sANum1, sANum2: string; inDist: integer): string;
```

Все действия над числами производятся так же в обратной записи, что значительно сокращает программу. Кратко работа алгоритма представляется следующими пунктами:

Приписываем ко второму слагаемому столько нулей, на сколько разрядов требуется сместить второе слагаемое относительно первого.

Находим, которое из двух слагаемых имеет большую длину.

Организуем цикл по длине наибольшего слагаемого и производим сложение посимвольно чисел, находящихся на одинаковых местах в строке. При этом результат может оказаться однозначным и двухзначным. Перенос двузначных чисел в следующий разряд совершенно аналогичен переносу цифр в алгоритме с умножением цифры на число.

Третья вспомогательная функция объединяет в себе возможности первых двух функций. Ниже она приведена полностью:

```
// УМНОЖЕНИЕ МНОГОЗНАЧНОГО ЧИСЛА НА МНОГОЗНАЧНОЕ ЧИСЛО
// ВХОД:  числа (обратный порядок)
// ВЫХОД: строка результата (обратный порядок)
function  multiplay(sNumber1, sNumber2: string): string;
var
  i: integer;
  sSum: string;
begin
  result := '';
  sSum := '0';
  for i := 1 to length(sNumber1) do
  begin
    // умножение
    result := multy(sNumber1[i],sNumber2);
    // сложение
    result := add(sSum, result, i – 1);
    sSum := result;
  end;
end;
```

Работа функции не требует пояснений и понятна из контекста. Она берёт два числа произвольной длины и возвращает их произведение.

В основной части программы запускается цикл от 3 до числа, факториал которого необходимо вычислить. И производится умножение результата (факториал двойки равен двум) на следующее число. В конечном итоге на экран выдаётся факториал искомого числа.

Факториал даже небольших чисел весьма громоздкое число, и чтобы найти факториал некоего числа, алгоритму приходится

вычислять и все предыдущие факториалы. Единственным преимуществом описанного алгоритма является нахождение точного значения факториала довольно больших чисел, надо только запастись терпением.

Второй алгоритм

Второй алгоритм работает гораздо быстрее, но позволяет находить только последнюю ненулевую цифру факториала. Он непригоден для точного нахождения всех цифр факториала. Прежде чем рассматривать сам алгоритм, полезно познакомится с математической теорией метода.

Факториал некоего числа n можно представить в виде [1]:

$$n! = (2m+1)\times 2^a, \text{ где } a > b$$

То есть, как произведение некоего нечётного числа и степеней двойки и пятёрки. В приведённой формуле: a – количество 2-к в числах от 2 до n, b – количество 5-к в числах от 2 до n.

Введём функцию $f(n)$, которая означает последнюю ненулевую цифру целого числа n. Например

$$f(12457852100000) = 1.$$

С помощью этой функции можно записать:

$$f(n!) = f((2m+1)\times 2^{(a-b)}\times 10^b) = f((2m+1)\times 2^{(a-b)})$$

Рассмотрим отдельно степени двойки. Известно, что последняя цифра степеней двойки образует цикл 2, 4, 8, 6. Таким образом, в зависимости от остатка $(a-b)/4$ можно вычислить функцию $f(2^{(a-b)})$.

Теперь рассмотрим последние цифры нечётных чисел. Нечётное число в нашем случае может оканчиваться на следующие цифры 1, 3, 7, 9. Рассматривать единицу не имеет смысла. Для остальных цифр существуют следующие циклы:

Для 3-ки – 3, 9, 7, 1;

Для 7-ки – 7, 9, 3, 1;

Для 9-ки – 9, 1;

Таким образом, задача нахождения последней ненулевой цифры факториала сводится к следующей формуле:

$$f(n!) = f(f(2^{(a-b)})\times f(3^{k_1})\times f(7^{k_2})\times f(9^{k_3}))$$

где k_1 – количество 3-к, k_2 – количество 7-к, k_3 – количество 9-к.

Из приведённых выше выкладок вытекает следующий алгоритм:

Необходимо подсчитать общее количество 2-к, 3-к, 5-к, 7-к и 9-к во всех числах от 2 до n включительно;

Далее вычисляем остатки от деления найденных чисел на соответствующие значения периодов;

По остаткам легко найти соответствующие цифры в последней формуле;

Последняя цифра произведения полученных цифр даст нам искомую последнюю ненулевую цифру факториала n.

Рассмотренный алгоритм получает своё дальнейшее развитие и логическое завершение в третьем алгоритме.

Третий алгоритм

Следующий алгоритм проще всего рассмотреть на основе конкретного примера. Определим последнюю ненулевую цифру числа 697!

Рассмотрим произведения последовательных десяток чисел:

1×2×3×4×**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**,

…

681×682×683×684×**685**×686×687×888×689×**690**,

691×692×693×694×**695**×696×697.

В этих произведениях будем отдельно рассматривать произведения чисел, кратных пяти (они выделены жирным шрифтом), и произведения остальных чисел. Здесь можно сделать следующее наблюдение, ключевое для всех последующих рассуждений: произведение невыделенных цифр в каждой десятке даёт в конце шестёрку. Произведение любого количества шестёрок будет опять давать в конце шестёрку. Поэтому можно в каждой десятке заменить все невыделенные числа на цифру шесть, без вреда для последующих рассуждений.

Исходя из сделанных наблюдений, заключаем, что искомая последняя цифра факториала будет определяться произведением:

$$f(697!) = (6\times 6\times 6\times\ldots\times 6)\times(\mathbf{5}\times\mathbf{10}\times\mathbf{15}\times\mathbf{20}\times\ldots\times\mathbf{680}\times\mathbf{685}\times\mathbf{690}\times\mathbf{695})\times(691\times 692\times 693\times 694\times 696\times 697);$$

Последняя цифра произведения в третьих скобках совпадает с последней цифрой произведения 1×2×3×4×6×7. Таким образом, задача сводится к нахождению последней ненулевой цифры во вторых скобках.

Определим функцию $g(n)$ для $n<10$, дающую последнюю ненулевую цифру произведения n последовательных цифр с исключением из произведения пятёрки:

$$g(n)=\left(\frac{1\times2\times3\times4\times6\times...\times n}{10}\right).$$

Круглые скобки означают дробную часть.
Тогда можно написать:

$g(0)=1$ (так как 0!=1); $g(1)=\left(\frac{1}{10}\right)=1$; $g(2)=\left(\frac{1\times2}{10}\right)=2$; $g(3)=\left(\frac{1\times2\times3}{10}\right)=3$;

$$g(4)=\left(\frac{1\times2\times3\times4}{10}\right)=4;$$

$g(5)=g(4)=4$; $g(6)=\left(\frac{1\times2\times3\times4\times6}{10}\right)=4$; $g(7)=\left(\frac{1\times2\times3\times4\times6\times7}{10}\right)=8$;

$$g(8)=\left(\frac{1\times2\times3\times4\times6\times7\times8}{10}\right)=4;\ g(9)=\left(\frac{1\times2\times3\times4\times6\times7\times8\times9}{10}\right)=6;$$

Так как 2×5=10 и эти произведения ничего не меняют в определении последней ненулевой цифры, исключаем из произведения скобках все пятёрки и, за одно, исключаем столько же двоек. Количество пятёрок в n! можно определить по известной формуле [1]:

$$k=\left[\frac{n}{5}\right]+\left[\frac{n}{5^2}\right]+\left[\frac{n}{5^2}\right]+\ldots+\left[\frac{n}{5^j}\right],\ \text{где}\ 5^j\le n;$$

здесь k – количество пятёрок, $\left[\frac{a}{b}\right]$ – целая часть деления a на b.

После исключения из рассмотрения k двоек последняя цифра факториала будет определяться произведением:

$$f(697!)=(6)\times((3\times3\times\ldots\times3)\times(\mathbf{5\times10\times15\times20\times\ldots\times680\times690\times695}))\times g(7),$$

или

$$f(697!)=(6)\times((3^k)\times(\mathbf{10\times15\times20\times\ldots\times680\times690\times695}))\times g(7);$$

Чтобы определить последнюю цифру для 3^k, определим функцию d:

$$d(0)=\left(\frac{3^0}{10}\right)=1,\ d(1)=\left(\frac{3^1}{10}\right)=3,$$

и т.п. $d(2)=9$, $d(3)=7$ (далее эти значения будут периодически повторятся). Если обозначить $c=\left(\frac{k}{4}\right)$, то нас будет интересовать число $d(c)$; тогда последняя ненулевая цифра в искомом факториале может определяться следующим произведением:

$$f(697!)=(6)\times(d(c)\times(\mathbf{5\times10\times15\times20\times\ldots\times680\times690\times695}))\times g(7);$$

Теперь рассмотрим числа, которые кратный 5:
Разделив каждое число во вторых скобках на 5, получим следующее произведение:

$$f(697!)=6\times d(c)\times(\mathbf{5/5\times10/5\times15/5\times20/5\times25/5\times\ldots\times675/5\times680/5\times685/5\times690/5\times695/5})\times g(7);$$

$$f(697!)=6\times d(c)\times(1\times2\times3\times4\times\mathbf{5}\times\ldots\times\mathbf{135}\times136\times137\times138\times139)\times g(7);$$

Снова пропуская числа, кратные 5, получим:

$$f(697!)=6\times d(c)\times(g(9)\times g(9)\times\ldots\times g(9)\times\mathbf{5}\times\mathbf{10}\times\ldots\times\mathbf{135)}\times g(7);$$

Снова разделим числа, кратные пяти на пять и получим:

$$f(697!)=6\times d(c)\times (g(9)\times g(9)\times\ldots\times g(9)\times\mathbf{5/5}\times\mathbf{10/5}\times\ldots\times\mathbf{135/5)}\times g(7);$$

$$f(697!)=6\times d(c)\times (g(9)\times g(9)\times\ldots\times g(9)\times 1\times 2\times 3\times 4\times\mathbf{5}\times\ldots\times\mathbf{25}\times 26\times 27\mathbf{)}\times g(7);$$

$$f(697!)=6\times d(c)\times (g(9)\times g(9)\times\ldots\times g(9)\times g(7)\times\mathbf{5}\times\ldots\times\mathbf{25)}\times g(7);$$

Ещё раз делим на пять:

$$f(697!)=6\times d(c)\times (g(9)\times g(9)\times\ldots\times g(9)\times g(7)\times\mathbf{5/5}\times\ldots\times\mathbf{25/5)}\times g(7);$$

$$f(697!)=6\times d(c)\times (g(9)\times g(9)\times\ldots\times g(9)\times g(7)\times 1\times 2\times 3\times 4\times\mathbf{5)}\times g(7);$$

$$f(697!)=6\times d(c)\times (g(9)\times g(9)\times\ldots\times g(9)\times g(7)\times g(5)\mathbf{)}\times g(7);$$

$$f(697!)=6\times d(c)\times (g(9)\times g(7)\times g(5))\times g(7);$$

Для нашего примера

$k=172;\ c=0;\ d(0)=1;$

Окончательно получим:

$$f(697!)=6\times 1\times(6\times 8\times 4)\times 8=9216.$$

Последняя цифра найденного числа даст нам последнюю ненулевую цифру факториала от 697.

Пример реализации рассмотренного алгоритма приведен ниже:

```
Uses crt;
function k(n: longint): longint;
var x, s: longint;
begin
  x:= 5; s:= 0;
   while x<= ndo
      begin
       s:= s+ trunc(n/x);
       x:= x*5;
      end;
 k:= s;
end;
function g(n: longint): longint;
 begin
  case nof
    0: g:= 1; 1: g:= 1;  2: g:= 2;  3: g:= 6;  4: g:= 4;
    5: g:= 4; 6: g:= 4;  7: g:= 8;  8: g:= 4;  9: g:= 6; 10: g:= 6;
  end;
 end;
function d(n: longint): longint;
 begin
  case nmod 4 of
    0: d:= 1; 1: d:= 3;  2: d:= 9; 3: d:= 7;
  end;
 end;
 var n, r, p, c: longint;
Begin clrscr;
 writeln('n-?'); readln(n);
   r:= 1; c:= k(n);
   if (n = 0) or (n = 1) then r:= 1
                  else
     begin
       while n> 0 do
```

```
        begin
          p:= nmod 10;
            r:= (r*g(p))mod 10;
          n:= ndiv 5;
        end;
       r:= (6*r*d(c))mod 10;
    end; writeln(r);
  readkey;
end.
```

Заключение

Все три алгоритма были проверены на быстродействие. Как и ожидалось, первый алгоритм показал самое большое время работы. За приемлемое время здесь можно вычислить факториалы чисел не больших 100000. Его единственным достоинством является возможность вычисления точного значения факториала

Второй алгоритм факториал 100000 вычислил за 15 миллисекунд, факториал 1000000 вычислялся 108 миллисекунд, всё ещё приемлемое время. Но вот на вычисление 1000000000! уже потребовалось весьма значительное время – 143 секунды.

Время работы третьего алгоритма измерить так и не удалось. 2100000000! вычислялся за 0 миллисекунд. Использование больших чисел затруднялось ограничением типа данных.

Список литературы

1. Бугаенко В.О.. Турниры им. Ломоносова. Конкурсы по математике. – МЦНМО-ЧеРо, 1998.

2. Уоррен Генри С., мл. Глава 16. Формулы для простых чисел // Алгоритмические трюки для программистов – М.: «Вильямс», 2007. – 288 с.

3. Цагер Д. Первые 50 миллионов простых чисел // Успехи математических наук. – 1984. – Т. 39. – № 6(240). – С. 175–190.