Math-Net.Ru

Общероссийский математический портал

А. В. Ладиков, Улучшенный алгоритм вычисления факториала, *Матем. заметки*, 2008, том 83, выпуск 6, 857–863

DOI: <https://doi.org/10.4213/mzm4837>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением
<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 3.81.55.215

26 апреля 2023 г., 13:30:57

Улучшенный алгоритм вычисления факториала

А. В. Ладиков

В работе рассматривается метод улучшения алгоритма вычисления факториалов целых чисел, позволяющий линейно ускорить вычисления, приводятся алгоритмы эффективно реализующие данный метод, анализируется сложность приведенных алгоритмов.

Библиография: 2 названия.

1. Введение. Факториал целого неотрицательного числа n определяется как произведение всех целых чисел от 1 до n , причем, факториал нуля будет равен единице:

$$n! = 1 * 2 * 3 * \dots * (n - 1) * n.$$

Чтобы не касаться вопросов труднорешаемости задачи вычисления факториалов целых чисел, связанных с экспоненциальным ростом длины конечного результата при незначительном увеличении входного параметра n , введем следующее упрощение: будем рассматривать вычисление факториала по некоторому модулю p , причем: $p > n$, p – простое число.

Исходя из данного определения, формулируется следующий алгоритм нахождения факториала числа n .

АЛГОРИТМ А. Простейший алгоритм вычисления факториала:

- 1) подать на вход алгоритма число n ;
- 2) установить значение переменной Result равное единице;
- 3) для всех целых I от 1 до n выполнить: $\text{Result} \leftarrow (\text{Result} * I) \bmod p$;
- 4) выдать результат (переменная Result).

Данный алгоритм производит n операций умножения в одном цикле от 1 до n . Следовательно, алгоритм полиномиальный и его временная сложность $t(n) = M * n$, где M – константа, равная единице. Сложность данного алгоритма в асимптотическом приближении составляет $O(n)$. Существует ли алгоритм с асимптотической сложностью меньшей, чем $O(n)$, либо с существенно уменьшенным значением константы M (уменьшено количество повторений цикла) [1]? Далее приводится ранее не рассматриваемый способ уменьшения константы M , интересный с точки зрения прикладных вычислений.

2. Оптимизация алгоритма вычисления факториала. В основе рассматриваемого метода оптимизации алгоритма вычисления факториала лежит следующая

ТЕОРЕМА. Факториал любого целого неотрицательного числа n можно представить в виде

$$n! = 2^{\lfloor n/2 \rfloor} * \left\lfloor \frac{n}{2} \right\rfloor! * n!!!,$$

где $n!!!$ – факториал по нечетным числам, а $\lfloor x \rfloor$ – целая часть числа x .

ДОКАЗАТЕЛЬСТВО. Факториал любого числа можно представить в следующем виде: $n! = n!! * n!!!$, где $n!!$ – факториал по четным числам, а $n!!!$ – факториал по нечетным числам. Рассмотрим факториал по четным числам:

$$n!! = 2 * 4 * 6 * 8 * \dots * \begin{cases} n, & \text{если } n \text{ четное,} \\ (n-1), & \text{если } n \text{ нечетное;} \end{cases}$$

рассмотрим теперь $(n/2)!$:

$$\left(\frac{n}{2}\right)! = 1 * 2 * 3 * \dots * \left\lfloor \frac{n}{2} \right\rfloor;$$

предположим, что n четное, и разделим $n!!$ на $(n/2)!$:

$$\frac{n!!}{(n/2)!} = \frac{2 * 4 * 6 * 8 * \dots * n}{1 * 2 * 3 * 4 * \dots * (n/2)} = 2 * 2 * 2 * \dots * 2 = 2^{n/2}.$$

Следовательно, любой факториал по четным числам можно представить в виде

$$n!! = 2^{\lfloor n/2 \rfloor} * \left\lfloor \frac{n}{2} \right\rfloor!,$$

тогда

$$n! = n!! * n!!! = 2^{\lfloor n/2 \rfloor} * \left\lfloor \frac{n}{2} \right\rfloor! * n!!!.$$

Рекурсивно применяя данный метод, получаем следующее разложение числа $n!$:

$$n! = n!! * n!!! = 2^{n_2+n_3+\dots+n_{k-1}+n_k} * n_k! * n_{k-1}!!! * \dots * n_2!!! * n_1!!!,$$

где

$$n_k = 1, \quad n_1 = n, \quad n_i = \left\lfloor \frac{n_{i-1}}{2} \right\rfloor, \quad i \neq 1.$$

Введем следующее упрощение записи: пусть $[a_j * a_{j+1} * \dots * a_k]$ есть произведение всех нечетных чисел, лежащих в интервале от a_j до a_k . Например, $[2 * 3 * 4 * 5 * 6 * 7 * 8 * 9] = (3 * 5 * 7 * 9)$. Тогда $n_i!!!$ можно представить как

$$n_i!!! = [1 * 3 * 5 * \dots * n_{i+1} * \dots * n_i] = n_{i+1}!!! * [(n_{i+1} + 1) * \dots * n_i], \quad i \neq k;$$

тогда значение $n!$ вычисляется по следующей формуле:

$$n! = 2^{n_2+n_3+\dots+n_{k-1}+n_k} * n_k! * n_{k-1}!!! * \underbrace{[n_{k-1}!!! * \dots * n_{k-2}]}_{n_{k-2}!!!} * \underbrace{[n_{k-2}!!! * \dots * n_{k-3}]}_{n_{k-3}!!!} * \dots * \underbrace{[n_2!!! * \dots * n_1]}_{n_1!!!}. \quad (1)$$

При этом количество разложений, необходимых для получения конечного результата ($n_i = 1$), определяется как количество целочисленных делений n на два и равно $\lfloor \log_2(n) \rfloor$.

Выражение (1) служит основой для построения метода вычисления факториала с использованием следующего алгоритма (алгоритма разложения): пока $n \neq 1$ целочисленно делить n на 2, запоминая промежуточные результаты: $n_1 = n$, $n_2 = \lfloor n_1/2 \rfloor$, $\dots$, $n_i = \lfloor n_{i-1}/2 \rfloor$, $\dots$, $n_k = 1$. Начиная с n_{k-1} вычислить все $n_i!!!$, $(k-1) \leq i \leq 1$, используя при вычислении $n_{i-1}!!!$ значение $n_i!!!$, вычисленное в предыдущей итерации. Перемножить полученные значения $n_i!!!$ между собой, возвести число 2 в степень $n_2 + n_3 + \dots + n_i + \dots + n_k$ и домножить на результат перемножения всех $n_i!!!$, данное значение и есть искомое $n!$.

3. Анализ сложности алгоритма разложения. Проанализируем сложность приведенного алгоритма. Для этого рассмотрим вычисление факториала по формуле (1). Как видно, алгоритм разбивает задачу вычисления факториала на три задачи: вычисление каждого $n_i!!!$, $k < i < 1$, вычисление 2^q , вычисление произведений между всеми n_i и умножение полученного результата на 2^q , $q = n_2 + n_3 + \dots + n_{k-1} + n_k$.

Для вычисления 2^q существует асимптотически “быстрый” алгоритм со сложностью $t_1(q) = O(\log_2(q))$, где q – степень числа 2 [2]. Но так как

$$q = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \dots + \left\lfloor \frac{n}{2^{\lfloor \log_2(n) \rfloor}} \right\rfloor,$$

то $q \approx n$. Тогда

$$t_1(q) = t_1(n) = O(\log_2(n)).$$

Рассмотрим вычисление $n_i!!!$. Так как для вычисления каждого $n_i!!!$ используются только нечетные числа, то для вычисления $n_i!!!$ потребуется произвести $n_i/2$ операций. Если вычислять $n_i!!!$ как $[n_{i+1}!!! * (n_{i+1} + 1) * \dots * n_i]$, то половина операций производится в предыдущем вычислении $n_{i+1}!!!$, так как $n_{i+1} = \lfloor n_i/2 \rfloor$. Тогда для вычисления $n_i!!!$ потребуется произвести $n_i/4$ операций. Значит, для оценки сложности вычисления всех $n_i!!!$ и их произведений необходимо воспользоваться следующим рекурсивным уравнением:

$$T(n) = \begin{cases} 1, & \text{если } n = 1, \\ T\left(\frac{n}{2}\right) + \frac{n}{4} + 1, & \text{если } n > 1, \end{cases}$$

где слагаемое $T(n/2)$ отражает следующую рекурсию, $n/4$ – количество операций необходимых для вычисления текущего разложения, единица отражает умножение полученного результата на результат, вычисленный на предыдущем шаге. Решим данное уравнение:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + \frac{n}{4} + 1 = T\left(\frac{n}{4}\right) + \frac{n}{8} + \frac{n}{4} + 1 + 1 \\ &= T\left(\frac{n}{8}\right) + \frac{n}{16} + \frac{n}{8} + \frac{n}{4} + 1 + 1 + 1 = \dots \\ &= T(1) + \frac{n}{2^{\lfloor \log_2(n) \rfloor}} + \dots + \frac{n}{16} + \frac{n}{8} + \frac{n}{4} + (1 + 1 + \dots + 1). \end{aligned}$$

Так как

$$\left(\frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots\right) = \frac{1}{2} * \left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots\right),$$

и

$$\left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots\right)$$

есть ряд, сумма элементов которого стремится к n при $n \rightarrow \infty$, то

$$\frac{1}{2} * \left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots\right)$$

не превзойдет $n/2$. Сумма $(1 + 1 + \dots + 1)$ есть $\lfloor \log_2(n) \rfloor$, так как производится $\lfloor \log_2(n) \rfloor$ входов в рекурсию, $T(1) = 1$. Тогда решение данного уравнения имеет вид

$$T(n) = \frac{n}{2} + \lfloor \log_2(n) \rfloor + 1.$$

Сложность алгоритма разложения определяется как сумма сложностей алгоритмов: вычисления степени двойки, вычисления разложения каждого числа $n_i!!!$, вычисления произведений между $n_i!!!$ и $n_{i+1}!!!$ (включено в решение рекурсивного уравнения). Тогда алгоритм разложения имеет следующую сложность:

$$t(n) = T(n) + O(\log_2(n)) = \frac{n}{2} + \lfloor \log_2(n) \rfloor + 1 + O(\log_2(n)) = \frac{n}{2} + O(\log_2(n)).$$

Данный алгоритм предполагает только константное улучшение, так как $n/2 + O(\log_2(n)) = O(n)$. При этом константа улучшения сложности алгоритма по сравнению с алгоритмом А будет стремиться к $1/2$ при $n \rightarrow \infty$:

$$\lim_{n \rightarrow \infty} \frac{n/2 + O(\log_2(n))}{n} = \frac{1}{2}.$$

4. Описание алгоритма.

АЛГОРИТМ В. Улучшенный алгоритм вычисления факториала:

- 1) подать на вход алгоритма значение n , факториал которого будет вычислен;
- 2) установить начальные значения переменных: $\text{Result} \leftarrow 1$; $\text{Buf} \leftarrow 1$; $\text{Stepen} \leftarrow 1$; Result – переменная результата, Buf – промежуточный результат, Stepen – переменная для хранения степени двойки; все переменные – большие целочисленные типы;
- 3) выделить стек; требуемый размер стека – $\lfloor \log_2(n) \rfloor$;
- 4) если $n = 0$, перейти к п. 17);
- 5) если $n = 1$, перейти к п. 9);
- 6) поместить текущее значение n в стек;
- 7) получить новое значение n , $n \leftarrow \lfloor n/2 \rfloor$;
- 8) увеличить значение степени двойки на n , $\text{Stepen} \leftarrow \text{Stepen} + n$; перейти к п. 5);
- 9) проверить стек на наличие в нем данных; если стек пуст, перейти к п. 15);
- 10) получить из стека новое значение переменной n_1 ;
- 11) если $n > n_1$, перейти к п. 14);
- 12) умножить значение переменной Buf на текущее значение переменной n , $\text{Buf} \leftarrow \text{Buf} * n$;

- 13) увеличить значение n на два, получив следующее нечетное число; вернуться к п. 11);
- 14) умножить значение переменной Result на полученное значение переменной Buf, $\text{Result} \leftarrow \text{Result} * \text{Buf}$; перейти к п. 9);
- 15) поместить в Buf вычисленное значение степени двойки, $\text{Buf} \leftarrow 2^{\text{Stepen}}$;
- 16) умножить значение переменной Result на полученное значение переменной Buf, $\text{Result} \leftarrow \text{Result} * \text{Buf}$;
- 17) выдать результат (значение Result).

5. Алгоритм разложения, не требующий стековой памяти. Приведенный алгоритм имеет недостаток: для вычисления факториала требуется логарифмический стек, что может быть неудобно при очень больших значениях n . Данный недостаток можно устранить следующим способом: начать вычисления разложения не с последнего полученного $n_i!!!$, а с первого. Рассмотрим вычисление $n!$ по формуле (1) и преобразуем его следующим образом:

$$\begin{aligned}
 n! &= 2^q * n_k! * n_{k-1}!!! * n_{k-2}!!! * \dots * n_i!!! * \dots * n_2!!! * n_1!!! \\
 &= 2^q * 1! * [1 * 3 * \dots * n_{k-1}] \\
 &\quad * [1 * 3 * \dots * n_{k-1} * \dots * n_{k-2}] * \dots \\
 &\quad * [1 * 3 * \dots * n_{k-1} * \dots * n_{k-2} * \dots * n_i] * \dots \\
 &\quad * [1 * 3 * \dots * n_{k-1} * \dots * n_{k-2} * \dots * n_i * \dots * n_2] \\
 &\quad * [1 * 3 * \dots * n_{k-1} * \dots * n_{k-2} * \dots * n_i * \dots * n_2 * \dots * n_1] \\
 &= 2^q * [1 * 3 * \dots * n_{k-1}]^{k-1} * [(n_{k-1} + 1) * \dots * n_{k-2}]^{k-2} \\
 &\quad * [(n_{k-2} + 1) * \dots * n_{k-3}]^{k-3} * \dots * [(n_{i+1} + 1) * \dots * n_i]^i * \dots \\
 &\quad * [(n_3 + 1) * \dots * n_2]^2 * [(n_2 + 1) * \dots * n_1]^1.
 \end{aligned}$$

Как видно, для вычисления $n!$ данным способом необходимо вычислить каждое выражение вида $[(n_{i+1}+1) * \dots * n_i]$ и возвести его в степень, соответствующую номеру очередного произведенного разложения, начиная нумерацию с единицы. Функция сложности данного алгоритма имеет вид

$$t(n) = T(n) + O(\log_2(n)),$$

где $O(\log_2(n))$ – сложность вычисления степени двойки, а $T(n)$ – рекурсивная функция следующего вида:

$$T(n) = \begin{cases} 1, & \text{если } n = 1, \\ T\left(\frac{n}{2}\right) + \frac{n}{4} + 1 + W(\text{Inc}(x)), & \text{если } n > 1, \end{cases}$$

где $\text{Inc}(x)$ – означает увеличение значения переменной x с каждым новым входом в рекурсивную процедуру (x – степень, в которую необходимо возвести очередное вычисляемое разложение). Функция W отражает сложность возведения в степень x очередного вычисленного разложения. При этом функция W обладает свойствами логарифмической функции: $W(x) + W(y) = W(x * y)$, так как для возведения числа

в целую положительную степень используется “быстрый” алгоритм со сложностью $O(\log_2(x))$, $W(x) = O(\log_2(x))$. Найдем решение данного уравнения:

$$\begin{aligned}
 T(n) &= T\left(\frac{n}{2}\right) + \frac{n}{4} + W(1) + 1 \\
 &= T\left(\frac{n}{4}\right) + \frac{n}{8} + \frac{n}{4} + W(2) + W(1) + 1 + 1 \\
 &= T\left(\frac{n}{8}\right) + \frac{n}{16} + \frac{n}{8} + \frac{n}{4} + W(3) + W(2) + W(1) + 1 + 1 + 1 = \dots \\
 &= T(1) + \frac{n}{2^{\lfloor \log_2(n) \rfloor}} + \frac{n}{2^{\lfloor \log_2(n) \rfloor - 1}} + \dots + \frac{n}{16} + \frac{n}{8} + \frac{n}{4} \\
 &\quad + W(\lfloor \log_2(n) \rfloor) + W(\lfloor \log_2(n) \rfloor - 1) + \dots \\
 &\quad + W(3) + W(2) + W(1) + 1 + 1 + \dots + 1 \\
 &= \frac{n}{2} + 1 + W(\lfloor \log_2(n) \rfloor * (\lfloor \log_2(n) \rfloor - 1) * \dots * 2 * 1) + \lfloor \log_2(n) \rfloor \\
 &= \frac{n}{2} + W(\lfloor \log_2(n) \rfloor!) + \lfloor \log_2(n) \rfloor + 1.
 \end{aligned}$$

Тогда сложность алгоритма разложения, не требующего стековой памяти, определяется следующим образом:

$$\begin{aligned}
 t(n) &= \frac{n}{2} + W(\lfloor \log_2(n) \rfloor!) + \lfloor \log_2(n) \rfloor + 1 + O(\log_2(n)) \\
 &= \frac{n}{2} + O(\log_2(\lfloor \log_2(n) \rfloor!)) + O(\log_2(n)).
 \end{aligned}$$

Оптимизация данного алгоритма, по сравнению с алгоритмом [A](#), также составляет $1/2$ при $n \rightarrow \infty$:

$$\lim_{n \rightarrow \infty} \frac{n/2 + O(\log_2(\lfloor \log_2(n) \rfloor!)) + O(\log_2(n))}{n} = \frac{1}{2} + \lim_{n \rightarrow \infty} \frac{O(\log_2(\lfloor \log_2(n) \rfloor!))}{n};$$

так как

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n,$$

то

$$\lfloor \log_2(n) \rfloor! \approx \sqrt{2\pi \lfloor \log_2(n) \rfloor} \left(\frac{\lfloor \log_2(n) \rfloor}{e}\right)^{\lfloor \log_2(n) \rfloor};$$

тогда

$$\begin{aligned}
 &\lim_{n \rightarrow \infty} \frac{\log_2(\lfloor \log_2(n) \rfloor!)}{n} \\
 &= \lim_{n \rightarrow \infty} \frac{\log_2(\sqrt{2\pi \lfloor \log_2(n) \rfloor} (\lfloor \log_2(n) \rfloor / e)^{\lfloor \log_2(n) \rfloor})}{n} \\
 &= \lim_{n \rightarrow \infty} \frac{\log_2(\sqrt{2\pi \lfloor \log_2(n) \rfloor}) + \lfloor \log_2(n) \rfloor * (\log_2 \lfloor \log_2(n) \rfloor - \log_2(e))}{n} = 0.
 \end{aligned}$$

Следовательно,

$$\frac{1}{2} + \lim_{n \rightarrow \infty} \frac{O(\log_2(\lfloor \log_2(n) \rfloor!))}{n} = \frac{1}{2}.$$

6. Описание алгоритма разложения, не требующего стековой памяти.

АЛГОРИТМ С. Улучшенный алгоритм вычисления факториала, не требующий организации стека:

- 1) получить значение n , факториал которого будет вычислен;
- 2) установить начальные значения переменных: $\text{Result} \leftarrow 1$, $\text{Buf} \leftarrow 1$, $\text{Stepen} \leftarrow 0$, $k \leftarrow 0$, $n_1 \leftarrow 0$, $n_2 \leftarrow 0$; Result – переменная хранения результата, Buf – переменная для хранения промежуточных результатов, Stepen – переменная степени двойки, n_1 , n_2 – промежуточные буферы, k – переменная для хранения степени текущего разложения;
- 3) если $n = 0$, перейти к п. 21);
- 4) если $n = 1$, перейти к п. 19);
- 5) присвоить переменной n_1 значение переменной n , $n_1 \leftarrow n$;
- 6) целочисленно разделить n на два, $n \leftarrow \lfloor n/2 \rfloor$;
- 7) присвоить переменной n_2 значение переменной n , $n_2 \leftarrow n$;
- 8) увеличить значение переменной Stepen на n , $\text{Stepen} \leftarrow \text{Stepen} + n$;
- 9) увеличить значение переменной n_2 на единицу, предотвратив повторное умножение на одно и то же число, $n_2 \leftarrow n_2 + 1$;
- 10) увеличить значение переменной k на единицу, $k \leftarrow k + 1$;
- 11) проверить n_2 на четность, если нечетное, перейти к п. 13);
- 12) увеличить значение n_2 на единицу, сделав его нечетным;
- 13) установить значение $\text{Buf} \leftarrow 1$;
- 14) если $n_2 > n_1$, перейти к п. 17);
- 15) умножить значение переменной Buf на значение переменной n_2 , $\text{Buf} \leftarrow \text{Buf} * n_2$;
- 16) увеличить значение переменной n_2 на два, $n_2 \leftarrow n_2 + 2$; перейти к п. 14);
- 17) возвести значение переменной Buf в степень k , $\text{Buf} \leftarrow \text{Buf}^k$;
- 18) умножить результат на значение переменной Buf , $\text{Result} \leftarrow \text{Result} * \text{Buf}$; перейти к п. 4);
- 19) вычислить значение степени двойки и поместить ее в переменную Buf , $\text{Buf} \leftarrow 2^{\text{Stepen}}$;
- 20) умножить результат на значение переменной Buf , $\text{Result} \leftarrow \text{Result} * \text{Buf}$;
- 21) выдать результат (значение Result).

7. Заключение. Рассмотренные методы улучшения алгоритма вычисления факториала дают лишь константное улучшение сложности, в то время как алгоритмы с асимптотическим улучшением до сих пор не найдены и их существование является открытой научной проблемой [1].

СПИСОК ЦИТИРОВАННОЙ ЛИТЕРАТУРЫ

- [1] Н. Н. Кузюрин, *Лекции по курсу “Сложность комбинаторных алгоритмов”*, препринт ИСПРАН, 2003.
- [2] Д. Э. Кнут, *Искусство программирования*, т. 2: Полужисленные алгоритмы, М., Изд. дом “Вильямс”, 2003.

А. В. Ладиков

Московский государственный университет путей сообщения
(МИИТ)

E-mail: AndreyNot@mail.ru

Поступило

21.11.2006