Иван Бессонов @ibessonov
Программист

сегодня в 00:03

Быстрое вычисление факториала — PrimeSwing

Математика*, Алгоритмы*

Наткнувшись недавно на эту статью, я понял, что редко упоминаются способы вычисления факториала, отличные от банального перемноже последовательных чисел. Нужно эту ситуацию исправить.
Предлагаю рассмотреть «асимптотически наиболее быстрый» алгоритм вычисления факториала!

Для начала напомню, что факториал n — это произведение всех натуральных чисел от 1 до n ($n! = 1 \cdot 2 \cdot \ldots \cdot n$), при этом $0! = 1$;

1. Декомпозиция факториала

Введём функцию, именуемую *swinging factorial*, следующим образом:

$$n\wr = \frac{n!}{\lfloor n/2 \rfloor !^2}$$

Данная дробь всегда будет целым числом по простой причине — она кратна центральному биномиальному коэффициенту $\binom{n}{\lfloor n/2 \rfloor}$, который ра

$$\binom{n}{\lfloor n/2 \rfloor} = \frac{n!}{\lfloor n/2 \rfloor ! \cdot \lceil n/2 \rceil !}$$

Разворачивая определение *swinging factorial*, мы получим новую рекуррентную формулу факториала:

$$n! = \lfloor n/2 \rfloor !^2 \cdot n\wr$$

Она будет особенно хороша, если мы научимся эффективно вычислять значения $n\wr$.

2. Простые множители *swinging factorial*

Обозначим $l_p(n\wr)$ как степень простого числа p в примарном разложении $n\wr$. Тогда будет справедлива следующая формула:

$$l_p(n\wr) = \sum_{k \geqslant 1} \lfloor \frac{n}{p^k} \rfloor \bmod 2$$

Доказательство

Воспользуемся теоремой Лежандра о простых множителях факториала:

$$\begin{aligned} l_p(n!/\lfloor n/2 \rfloor !^2) &= l_p(n!) - 2l_p(\lfloor n/2 \rfloor !) \\ &= \textstyle\sum_{k \geqslant 1} \lfloor n/p^k \rfloor - 2\sum_{k \geqslant 1} \lfloor \lfloor n/2 \rfloor / p^k \rfloor \\ &= \textstyle\sum_{k \geqslant 1} (\lfloor n/p^k \rfloor - 2\lfloor \lfloor n/p^k \rfloor / 2 \rfloor) \end{aligned}$$

Для последнего выражения воспользуемся тем фактом, что $j - 2\lfloor j/2 \rfloor = j \bmod 2$, и получим нужную нам формулу.

Как следствие, $l_p(n\wr) \leqslant log_p(n)$ и $p^{l_p(n\wr)} \leqslant n$. Если p нечётно, то $l_p(p^a\wr) = a$. Другие частные случаи:

$$\begin{array}{lrcll} (a) & \lfloor n/2 \rfloor & < p \leqslant & n & \Rightarrow \quad l_p(n\wr) = 1 \\ (b) & \lfloor n/3 \rfloor & < p \leqslant & \lfloor n/2 \rfloor & \Rightarrow \quad l_p(n\wr) = 0 \\ (c) & \sqrt{n} & < p \leqslant & \lfloor n/3 \rfloor & \Rightarrow \quad l_p(n\wr) = \lfloor n/p \rfloor \bmod 2 \\ (d) & 2 & < p \leqslant & \sqrt{n} & \Rightarrow \quad l_p(n\wr) < log_2(n) \\ (e) & & p = & 2 & \Rightarrow \quad l_p(n\wr) = \sigma_2(\lfloor n/2 \rfloor) \end{array}$$

$\sigma_2(n)$ здесь означает количество единиц в двоичном представлении числа n. Все эти факты могут быть использованы для дополнительной оптимизации в коде. Доказательства я приводить не буду, при желании вы легко сможете получить их самостоятельно.

Теперь, зная степени всех простых делителей $n\wr$, у нас есть способ вычисления *swinging factorial*:

$$n\wr = \prod_{p \leqslant n} p^{l_p(n\wr)}$$

3. Трудоёмкость алгоритма

Можно показать, что вычисление $n!$ имеет сложность $O(n(\log n)^2 \log\log n)$. Как ни странно, вычисление $n!$ имеет ту же сложность (в оценке используется алгоритм Шёнхаге-Штрассена, отсюда и такая интересная трудоёмкость; доказательства по ссылке в конце статьи).

Несмотря на то, что формально перемножение чисел от 1 до n имеет ту же трудоёмкость, алгоритм *PrimeSwing* на практике оказывается сами быстрым.
UPDATE: как было замечено в этом комментарии, тут я ошибся, перемножение чисел от 1 до n имеет большую трудоёмкость.

Ссылки и реализация

- страница с различными алгоритмами вычисления факториала;
- детальное описание алгоритма из статьи (и не только).

Реализация на Java

```
// main function
public static BigInteger factorial(int n) {
    return factorial(n, primes(n));
}

// recursive function with shared primes array
private static BigInteger factorial(int n, int[] primes) {
    if (n < 2) return BigInteger.ONE;
    BigInteger f = factorial(n / 2, primes);
    BigInteger ps = primeSwing(n, primes);
    return f.multiply(f).multiply(ps);
}

// swinging factorial function
private static BigInteger primeSwing(int n, int[] primes) {
    List<BigInteger> multipliers = new ArrayList<>();
    for (int i = 0; i < primes.length && primes[i] <= n; i++) {
        int prime = primes[i];
        BigInteger bigPrime = BigInteger.valueOf(prime);
        BigInteger p = BigInteger.ONE;
        int q = n;
        while (q != 0) {
            q = q / prime;
            if (q % 2 == 1) {
                p = p.multiply(bigPrime);
            }
        }
        if (!p.equals(BigInteger.ONE)) {
            multipliers.add(p);
        }
    }
    return product(multipliers, 0, multipliers.size() - 1);
}

// fast product for the list of numbers
private static BigInteger product(List<BigInteger> multipliers, int i, int j) {
    if (i > j) return BigInteger.ONE;
    if (i == j) return multipliers.get(i);
    int k = (i + j) >>> 1;
    return product(multipliers, i, k).multiply(product(multipliers, k + 1, j));
}

// Eratosthenes sieve
private static int[] primes(int upTo) {
    upTo++;
    if (upTo >= 0 && upTo < 3) {
        return new int[]{};
    }

    int length = upTo >>> 1;
    boolean sieve_bool[] = new boolean[length];
    for (int i = 1, iterations = (int) Math.sqrt(length - 1); i < iterations; i++) {
        if (!sieve_bool[i]) {
```

```
            for (int step = 2 * i + 1, j = i * (step + 1); j < length; j += step) {
                sieve_bool[j] = true;
            }
        }
    }

    int not_primes = 0;
    for (boolean not_prime : sieve_bool) {
        if (not_prime) not_primes++;
    }

    int sieve_int[] = new int[length - not_primes];
    sieve_int[0] = 2;
    for (int i = 1, j = 1; i < length; i++) {
        if (!sieve_bool[i]) {
            sieve_int[j++] = 2 * i + 1;
        }
    }

    return sieve_int;
}
```

primeswing, факториал, факторизация, простые числа

3,4k 27

Иван Бессонов @ibessonov карма 30,0 рейтинг 27,2
Программист
Github Telegram

ПОХОЖИЕ ПУБЛИКАЦИИ

23 сентября 2013 в 00:03
Фракталы в простых числах
+176 103k 461 33

30 августа 2013 в 16:29
О простых числах, криптографии и повреждениях мозга
+53 38,1k 125 64

5 февраля 2013 в 23:39
Найдено 48-е простое число Мерсенна
+85 66,8k 77 140

САМОЕ ЧИТАЕМОЕ

Сейчас | Сутки | Неделя | Месяц

Получил 1.2K звезд на GitHub с ужасной архитектурой. Как?
+22 52,6k 102 39

Реализация псевдо-3D в гоночных играх

+90 20,7k 213 16

Индексы в PostgreSQL — 1

+102 18,7k 384 47

Выбор MQ для высоконагруженного проекта

+30 15,1k 154 51

Алгоритм Джонкера-Волгенанта + t-SNE = супер-сила

+63 14,5k 184 2

Комментарии (15)

отслеживать новые: в почте в трекере

koldyr 29 апреля 2017 в 07:32

> Теперь, зная степени всех простых делителей n!, у нас есть способ вычисления swinging factorial.

С какой сложностью нам достается это знание?

ответить

ibessonov 29 апреля 2017 в 09:20

Простых чисел от 1 до n всего O(n / log n), вычисление l_p происходит за O(log n).
Сложностью же перемножения простых чисел в соответствующих степенях будет O(n $(\log n)^2$ log log n) — это уже не очевидная оценка, её доказательство е
второй ссылке.

ответить

koldyr 29 апреля 2017 в 11:30

Асимптотику распределения простых чисел я знаю.
Я правильно понимаю, что придется хранить или генерировать таблицу простых чисел от 2 до n/2?
Какова трудоемкость детерминированного алгоритма теста на простоту?

ответить

ibessonov 29 апреля 2017 в 11:39

Не совсем, понадобится таблица простых чисел от 2 до n.
Решето Эратосфена составляется за O(n * log log n), это меньше, чем время вычисления факториала. Отдельного способа проверки на простоту алго
требует.

ответить

koldyr 29 апреля 2017 в 11:49

Прекрасно. Как-то я упустил, что сложность решета ниже сложности вычисления факториала.

ответить

choupa 29 апреля 2017 в 11:31

Мне кажется, самый практичный метод вычисления факториала

array[0, 1, 2, 6, 24, 120, ...]

Простите за серость, а зачем нужно вычислять большие факториалы? Криптография? Комбинаторика? Ряды Тейлора? Собеседование в Гугл?

ответить

ibessonov 29 апреля 2017 в 11:35

Боюсь, что это вопрос не ко мне. Вероятнее всего различные алгоритмы вычисления факториала имеют соревновательный характер.
Хранить массив может быть очень затратно по памяти, особенно если нужны значения порядка 1000000! (вдруг они кому-то нужны). На практике я такого не
встречал и сам всегда хранил кэшированные значения в обычном массиве (во время решения задач по программированию).

ответить

mikhaelkh 29 апреля 2017 в 11:46 +

> Несмотря на то, что формально перемножение чисел от 1 до n имеет ту же трудоёмкость, алгоритм PrimeSwing на практике оказывается самым быстрым.

Предположим, что разделим числа на две половины, рекурсивно вычислим произведение и перемножим половины. Сложность у этого метода ~M(n*log(n))*log(
хуже ~M(n*log(n)) у PrimeSwing. Или как предлагается перемножать?

ответить

ibessonov 29 апреля 2017 в 12:12

Извиняюсь, действительно у меня в этом моменте ошибка. Вычисление с помощью PrimeSwing имеет трудоёмкость как у простого перемножение двух чисе порядка n!, а не вычисления n! стандартным способом. Сейчас внесу обновление в статью.
Спасибо!

ответить

koldyr 29 апреля 2017 в 11:53

Стоит отметить, что статья имеет практическую направленность, но согласно вики:
Метод Шёнхаге — Штрассена считался асимптотически быстрейшим методом с 1971 до 2007 годы, пока не был заявлен новый метод с лучшей оценкой сложн умножения.[3] На практике метод Шёнхаге — Штрассена начинает превосходить более ранние классические методы, такие как умножение Карацубы и алгорит — Кука (обобщение метода Карацубы), начиная с целых чисел порядка 10^10000-10^40000.
Таким образом становится актуальным вопрос о константе в оценке.

ответить

ibessonov 29 апреля 2017 в 12:23

Интересно вышло, алгоритм PrimeSwing был опубликован в 2008-м, автор уже мог знать о более оптимальном способе умножения чисел.
Если я верно понял, автор производил вычисления факториалов примерно до 1000000, то есть результаты были не настолько большими, чтобы использова Алгоритма Фюрера (если верить википедии). А так да, теоретическую оценку наверняка можно улучшить.
На практике я не заморачивался и использовал то умножение, какое было реализовано за меня в BigInteger.
Спасибо за замечание.

ответить

mikhaelkh 29 апреля 2017 в 12:41

Хорошо про умножение с практической точки зрения написано здесь

ответить

malishich 29 апреля 2017 в 16:39

Стать интересная, не знал что и здесь простые числа применяются. На практике мне не встречались задачи где нужно было вычислять просто факториалы чи больше 20-30 (например в uint64_t влезает 20, а вот 21! уже нет, зато в double влезает, хоть и с погрешностью). Встречались отношения (дроби) больших факто (комбинаторные сочетания Cmn для близких чисел и размещения Amn для сильно разных чисел), порядка 300-т. Но в этих задачах я решал просто разложени простые множители всех чисел произведения в числителе и знаменателе, подсчёту количества таких множителей в числителе и знаменателе, сокращению, и дальнейшему вычислению отношения (там обычно много чего сокращалось и части дроби «влазили» с малой погрешностью в тип double, но чаще вообще в u

ответить

MikailBag 29 апреля 2017 в 22:08 *(комментарий был изменён)*

Зачем считать для цешек факториалы, если есть треугольник паскаля?
Я могу оценить время на генерацию N строк как O(N^3).
В Вашем случае (N = 300) — пара минут максимум.

ответить

mikhaelkh 29 апреля 2017 в 21:41

В этом алгоритме мы по сути выделяем большой множитель, являющийся квадратом. То есть A[k] = A[k+1]^2 * B[k+1], A[0] = n!, для вычисления A[k] мы вычисл A[k+1], B[k+1]. Но мы можем выделить ещё больший множитель, оставляя в B[k+1] только произведение простых, входящие в A[k] нечётных степенях. Кажется должно работать ещё быстрее, хотя разница должна быть невелика.

ответить