Санкт-Петербургский Государственный Университет

Факультет Прикладной математики – процессов управления

Лабораторная работа по курсу «Алгоритмы и анализ сложности» на тему «Эмпирический анализ алгоритма»

Автор: Крылова Ольга

333 группа ФИИТ

Содержание отчета:

1. Краткое описание алгоритма.
2. Математический анализ алгоритма.
3. Характеристики входных данных.
4. Формальная постановка задачи и ее решение (реализация на языке Python).
5. Реализация генератора.
6. Вычислительный эксперимент.
7. Анализ полученных данных.
8. Список литературы.

Краткое описание алгоритма:

Натуральное число называется простым, если оно имеет только два различных делителя: единицу и само себя. Задача поиска простых чисел не дает покоя математикам уже очень давно. Долгое время прямого практического применения эта проблема не имела, но все изменилось с появлением криптографии с открытым ключом.
Решето Аткина — алгоритм нахождения всех простых чисел до заданного целого числа *N*. Он был создан А. О. Л. Аткином и Д. Ю. Бернштейном в 1999 году.
Заявленная авторами асимптотическая скорость работы алгоритма соответствует скорости лучших ранее известных алгоритмов просеивания, но в сравнении с ними решето Аткина требует меньше памяти.

Основная идея алгоритма состоит в использовании неприводимых квадратичных форм (представление чисел в виде $ax^2 + by^2$). Предыдущие алгоритмы в основном представляли собой различные модификации решета Эратосфена, где использовалось представление чисел в виде редуцированных форм (как правило, в виде произведения xy). Отдельный этап алгоритма вычеркивает числа, кратные квадратам простых чисел.

Математический анализ алгоритма:

Теорема. Пусть n — натуральное число, которое не делится ни на какой полный квадрат. Тогда:

1. если n представимо в виде $4k+1$, то оно просто тогда и только тогда, когда число натуральных решений уравнения $4x^2+y^2 = n$ нечетно.

2. если n представимо в виде $6k+1$, то оно просто тогда и только тогда, когда число натуральных решений уравнения $3x^2+y^2 = n$ нечетно.

3. если n представимо в виде $12k-1$, то оно просто тогда и только тогда, когда число натуральных решений уравнения $3x^2-y^2 = n$, для которых $x > y$, нечетно.

Доказательство, приведенное создателями алгоритма, здесь.

Объяснение:

- Все числа, равные (по модулю 60) 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, или 58, делятся на два и заведомо не простые. Все числа, равные (по модулю 60) 3, 9, 15, 21, 27, 33, 39, 45, 51, или 57, делятся на три и тоже не являются простыми. Все числа, равные (по модулю 60) 5, 25, 35, или 55, делятся на пять и также не простые. Все эти остатки (по модулю 60) игнорируются.
- Все числа, равные (по модулю 60) 1, 13, 17, 29, 37, 41, 49 или 53, имеют остаток от деления на 4 равный 1. Эти числа являются простыми тогда и только тогда, когда количество решений уравнения $4x^2 + y^2 = n$ нечетно и само число не кратно никакому квадрату простого числа (en:square-free integer).
- Числа, равные (по модулю 60) 7, 19, 31, или 43, имеют остаток от деления на 6 равный 1. Эти числа являются простыми тогда и только тогда, когда количество решений уравнения $3x^2 + y^2 = n$ нечетно и само число не кратно никакому квадрату простого.
- Числа, равные (по модулю 60) 11, 23, 47, или 59, имеют остаток от деления на 12 равный 11. Эти числа являются простыми тогда и только тогда, когда количество решений уравнения $3x^2 - y^2 = n$ нечетно и само число не кратно никакому квадрату простого.

На начальном этапе алгоритма решето Аткина представляет собой массив *A* размером *n*, заполненный нулями. Для определения простых чисел перебираются все x, y < n. Для каждой такой пары вычисляется $4x^2+y^2$, $3x^2+y^2$, $3x^2-y^2$ и значение элементов массива $A[4x^2+y^2]$, $A[3x^2+y^2]$, $A[3x^2-y^2]$ увеличивается на единицу. В конце работы алгоритма индексы всех элементов массива, которые имеют нечетные значения либо простые числа, либо квадраты простого числа. На последнем шаге алгоритма производится вычеркивание квадратов оставшихся в наборе чисел.

Из описания алгоритма следует, что вычислительная сложность решета Аткина и потребление памяти составляют **O(n)**.

Возможные оптимизации:
При использовании wheel factorization и сегментирования оценка сложности алгоритма снижается до O(n/loglogn), а потребление памяти до O(n).

Характеристики входных данных:

На вход подается натуральное число. Алгоритм должен найти все простые числа от двойки до этого числа включительно ($3 < n < 10^7$).
Единица измерения трудоемкости во время эксперимента - время (в секундах).
Способ измерения - генератор для запуска алгоритма на различных входных данных и дальнейшее конструирование графика зависимости времени от n.
Входные данные: 1000000 <= n <= 7000000 с шагом в 1000000.

Для достоверности измерений для каждого n алгоритм запускается 20 раз и берется среднее время работы.

Формальное описание алгоритма:

1. Числа 2, 3 рассматриваются, как заведомо простые.
2. Создаётся «решето» — список, сопоставляющий каждому натуральному числу *n* из диапазона [5; *limit*] флаг «простое\составное». Изначально все флаги устанавливаются в положение «составное».
3. Для очередного *n* из диапазона [5; *limit*] находится остаток от деления на 60:
    - 1, 13, 17, 29, 37, 41, 49, 53: уравнение — $4x^2 + y^2 = n$; **(1)**
    - 7, 19, 31, 43: уравнение — $3x^2 + y^2 = n$; **(2)**
    - 11, 23, 47, 59: уравнение — $3x^2 - y^2 = n$ (при $x > y$) **(3)**
4. Значение остатка, не равное ни одному из вышеуказанных, свидетельствует о том, что число *n* — составное и таким образом исключается из дальнейшего рассмотрения. Иначе, если соответствующее уравнение имеет нечетное число решений (пар натуральных значений *x* и *y*), *n* помечается в «решете» как «простое».
5. Процедура повторяется начиная с шага № 3, пока не будут перебраны все $n \leq limit$.
6. В заключение для каждого найденного простого *n* все значения, кратные его квадрату, помечаются в «решете» как «составные».

Реализация на языке Python:

Произведена небольшая оптимизация:

> *n* делится с остатком на 12 (а не на 60). Полученные значения 1 и 5 соответствуют случаю **(1)**, 7 — **(2)**, 11 — **(3)**, прочие — составному *n*.

```
import math

def sieve_of_atkin(limit):
    prime_array = []
    if limit == 2:
        prime_array = [2]
    else:
        prime_array = [2, 3]
        is_prime = [False for i in range(limit + 1)]

        sqr_lim = int(math.sqrt(limit))

        for i in range(1, sqr_lim + 1):
            x2 = i * i
            for j in range(1, sqr_lim + 1):
                y2 = j * j

                # (1)
                n = 4 * x2 + y2
```

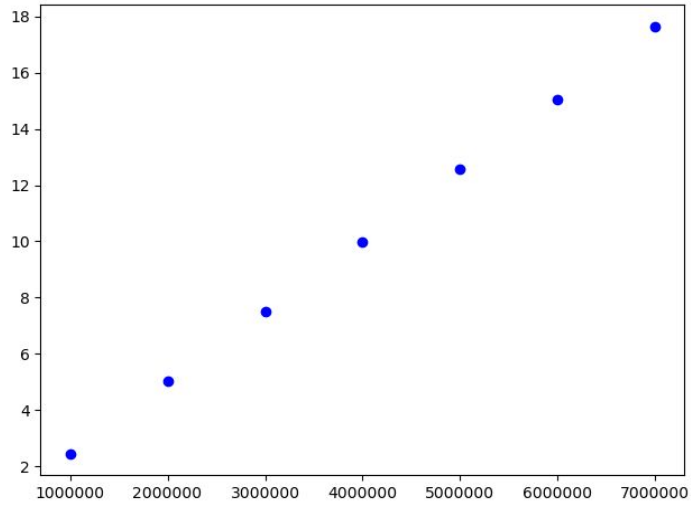
```
                if n <= limit and (n % 12 == 1 or n % 12 == 5):
                    is_prime[n] = not is_prime[n]

                # (2)
                n -= x2
                if n <= limit and n % 12 == 7:
                    is_prime[n] = not is_prime[n]

                # (3)
                n -= 2 * y2
                if i > j and n <= limit and n % 12 == 11:
                    is_prime[n] = not is_prime[n]

        for i in range(5, sqr_lim + 1, 2):
            if is_prime[i]:
                n = i * i
                for j in range(n, limit + 1, n):
                    is_prime[j] = False

        is_prime[2] = True
        is_prime[3] = True
        # for i in range(2, limit + 1):
        #    if is_prime[i]:
        #        prime_array.append(i)
        # print(prime_array, end='\n')
```

Реализация генератора для вычислительного эксперимента:

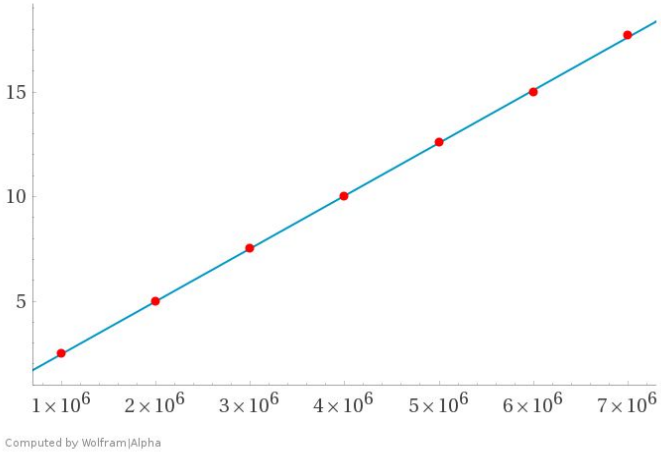
```
import matplotlib.pyplot as plt
import time

times = []
for k in range(1000000, 70000001, 1000000):
   times_sum = 0
   for i in range(20):
       start_time = time.time()
       sieve_of_atkin(k)
       times_sum += time.time() - start_time
   middle_time = times_sum / 10.0
   dots.append(k)
   times.append(middle_time)
   print(k, ': ', '{:.5f}'.format(middle_time), end='\n')


plt.scatter(dots, times, c='b')
plt.show()
```

Вычислительный эксперимент:

limit	time
1 000 000	2.45598
2 000 000	5.02680
3 000 000	7.49774
4 000 000	9.99584
5 000 000	12.57760
6 000 000	15.03600
7 000 000	17.65029



Аппроксимируем полученный график с помощью WolframAlpha:



Из графика можно увидеть, что функция зависимости времени от числа n зависит **линейно** => сложность алгоритма можно оценить как **O(n),** что соответствует заявленной теоретической сложности.

Список литературы:

1. https://ru.wikipedia.org/wiki/Решето_Аткина
2. https://habr.com/ru/post/125620/
3. https://habr.com/ru/post/468833/
4. https://en.wikipedia.org/wiki/Wheel_factorization
5. https://prog-cpp.ru/eratosfen/
6. https://www.ams.org/journals/mcom/2004-73-246/S0025-5718-03-01501-1/S0025-5718-03-01501-1.pdf