Лабораторная работа по курсу
«Алгоритмы и анализ сложности»
«Эмпирический анализ алгоритма решето Эратосфена»

Выполнила студентка третьего курса:
Сокол Милена, 17.Б11-пу

Содержание

Краткое описание алгоритма

Решето Эратосфена — это алгоритм генерации последовательности простых чисел, не превышающих произвольно заданного целого числа N. Вероятнее всего он был придуман в древней Греции и поэтому назван решетом Эратосфена (примерно 200 год до н.э.).

Алгоритм:
Для начала составим список, содержащий последовательность целых чисел от 2 до n, из которого затем мы должны будем выбрать простые числа. Далее, на первом проходе алгоритма нужно удалить из списка все числа, которые делятся на 2. Затем необходимо выбрать из списка следующий элемент (в данном случае это 3) и удалить из списка все числа, которые делятся на него. Следующим числом, которое осталось в списке и используется в третьем проходе, является 5. Работа алгоритма продолжается так до тех пор, пока в списке существуют числа, которые можно удалить. Числа, оставшиеся в списке после выполнения алгоритма, являются простыми.
При поиске кратных числу p нужно начинать рассмотрение с p*p, тк все кратные ему числа до этого были уже рассмотрены.

Источник: Левитин — Алгоритмы. Введение в разработку и анализ.

Математический анализ алгоритма

Оценим сложность алгоритма. Первое вычеркивание требует *n*/2 действий, второе — *n*/3, третье — *n*/5 и т. д. По формуле Мертенса

$$\sum_{\substack{p \le n \\ p \text{ простое}}} \frac{1}{p} \sim \log \log n,$$

так что для решета Эратосфена потребуется $O(n \log \log n)$ операций.

Более строгое доказательство дающее более точную оценку можно найти в книге Hardy и Wright «An Introduction to the Theory of Numbers».

Входные данные

В качестве входных данных возьмем произвольное N - мы ищем простые числа, ограниченные N сверху.
Будем рассматривать диапазон 500 000 ≤ N ≤ 5 000 000 с шагом в 500 000.
Единица измерения трудоемкости - время выполнения алгоритма (в секундах).

Генерация входных данных

Реализация написана на языке Python
На каждой итерации возвращает int_start ≤ N ≤ int_end с шагом step

```
def generator(int_start, int_end, step):
    if (int_start>int_end):
        print('Некорректные входные данные')
        exit()
    iter = int_start
    while iter<=int_end:
        yield iter
        iter += step
```

Программа реализации алгоритма

Написана на языке Python с использованием библиотеки Numpy

```
import numpy as np

def Eratosfen(N):
    bool_vec = np.ones(N+1)
    bool_vec[0:2] = 0
    i = 2
    while (i**2 <= N):
        if (bool_vec[i]):
            bool_vec[i**2:N+1:i] = 0
        i+=1
    return np.where(bool_vec)  #возвращает порядковые номера ненулевых элементов
```

Вычислительный эксперимент

Алгоритм вызывается для различных N несколько раз для получения усредненного значения времени выполнения, после чего строится график для анализа результатов.

```
def main():
    times = []
    for n in generator(500000,5000000,500000):
        sum_time = 0
        for _ in range(20):
            start_time = time.time()
            Eratosfen(n)
            sum_time += time.time() - start_time
        times.append(sum_time/20)

    plt.plot(np.arange(500000,5500000,500000), times)
    plt.title('Время выполнения алгоритма от N')
    plt.xlabel('N')
    plt.ylabel('Время (секунды)')
    plt.show()
```

Результаты эксперимента

N (число до которого ищем простые)	Время выполнения алгоритма (с)
500 000	0.014996
1 000 000	0.035789
1 500 000	0.054584
2 000 000	0.077777
2 500 000	0.101770
3 000 000	0.122764
3 500 000	0.140559
4 000 000	0.163249
4 500 000	0.179689
5 000 000	0.198341

Таблица: время выполнения алгоритма в зависимости от N

График построенный в результате проведения эксперимента:

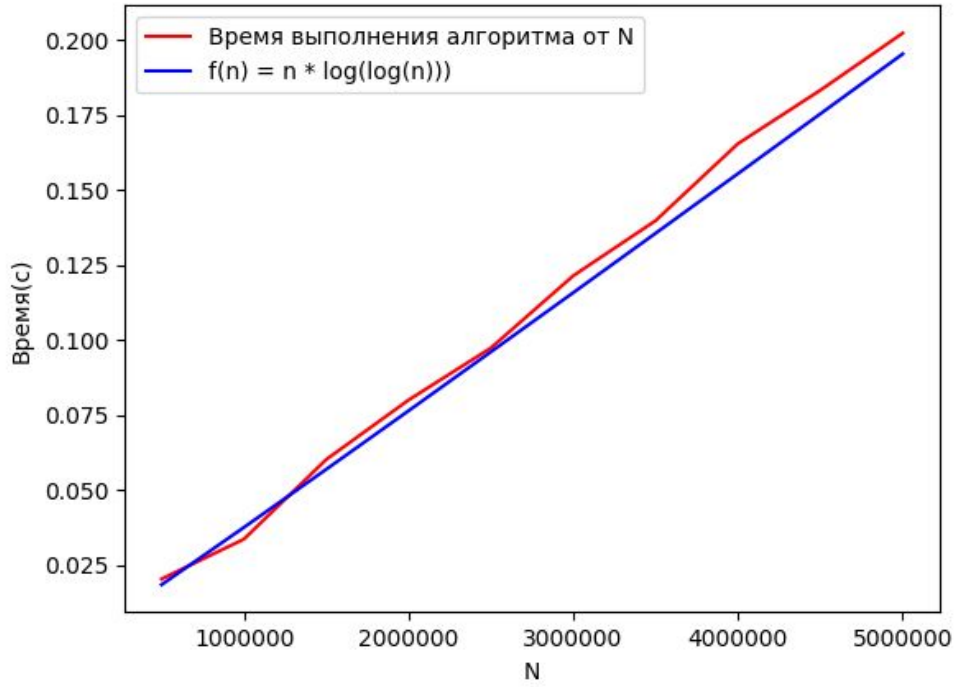

Рис: сравнительный график времени выполнения и теоретической сложности

С учетом погрешности мы можем сделать вывод, что теоретическая трудоемкость верна для нашей реализации.

Рассмотрим отношение значений измеренной трудоемкости при удвоении размера входных данных.

Для вычисления предела используем wolframalpha.com. Получаем результат:

$$\lim_{n\to\infty} \frac{2\,n\log(\log(2\,n))}{n\log(\log(n))} = 2$$

Проверим его практически, составив таблицу:

N1	N2 = 2*N1	T(2*N)/T(N)
500 000	1 000 000	2,3866
1 000 000	2 000 000	2,1732
1 500 000	3 000 000	2,2491
2 000 000	4 000 000	2,0989
2 500 000	5 000 000	1,9489

Таблица: отношение значений трудоемкости при удвоении N

По результатам мы видим, что данное отношение действительно стремится к 2 при N, стремящемся к бесконечности. То есть при достаточно больших N, если увеличивать N в два раза, время выполнения увеличится в два раза (время будет увеличиваться линейно).

Используемые источники:

1. Левитин — Алгоритмы. Введение в разработку и анализ.
2. Hardy и Wright «An Introduction to the Theory of Numbers»
3. https://habr.com/ru/post/133037/
4. https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%88%D0%B5%D1%82%D0%BE_%D0%AD%D1%80%D0%B0%D1%82%D0%BE%D1%81%D1%84%D0%B5%D0%BD%D0%B0

Реализация на github: https://github.com/tyugv/Sieve-of-Eratosthenes