Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
"Национальный исследовательский университет
"Высшая школа экономики"

Московский институт электроники и математики Национального исследовательского университета "Высшая школа экономики"

Лабораторная работа
по дисциплине
«Теоретико-числовые методы в криптографии»

Последовательности Лукса

Студент 1: Верендеев Кирилл Дмитриевич
Студент 2: Винников Никита
Преподаватель: Нестеренко Алексей Юрьевич

Группа: СКБ181

Москва
2022

Содержание

1 Введение

Тема данной работы - реализация теста на простоту, основываясь на свойствах последовательностей Лукаса и псевдопростых числах Лукаса.
Тест на простоту Лукаса может очень эффективно находить простые числа. Он делает это без ошибок до значения $25 * 10^9$, сложность всех вычислений составляет $O(log\ n)$, где n - число, проверяемое на простоту.

2 Теория

2.1 Последовательности Лукаса

Definition 2.1. Пусть D, P, Q - целые числа, такие что $D = P^2 - 4Q \neq 0, p > 0$. Пусть $U_0 = 0, U_1 = 1, V_0 = 2, V_1 = P$. Последовательность Лукаса U_k, V_k задается следующим образом для $k > 2$:

$$U_k = PU_{k-1} - QU_{k-2} \tag{1}$$

$$V_k = PV_{k-1} - QV_{k-2} \tag{2}$$

Тогда D - дискриминант характеристического многочлена $f(x) = x^2 - Px + Q$ данной рекуррентной последовательности.

Возьмем нечетное положительное n, обозначим $\sigma(n) = n - (\frac{D}{n})$, где $(\frac{D}{n})$ - символ Якоби. Тогда, если n - простое и если $(n, Q) = 1$, тогда

$$U_{\sigma(n)} = 0(mod\ n)) \tag{3}$$

2.2 Псевдопростые числа

Definition 2.2. Число называется псевдопростым по основанию a ($psp(a)$), если для n, где n - нечетное, выполняется

$$a^{n-1} = 1(mod\ n) \tag{4}$$

Definition 2.3. Число называется сильным псевдопростым по основанию a ($spsp(a)$), если для $n-1 = d2^s$, где d - нечетное, выполняется

$$\begin{aligned} &a^d = 1(mod\ n)\ or \\ &a^{d2^r} = 1(mod\ n), \exists r : 0 \leq r < s \end{aligned} \tag{5}$$

Definition 2.4. Число называется псевдопростым числом Лукаса с параметрами P, Q ($lpsp(P, Q)$), если для него выполняется условие (3), но оно является составным.

Definition 2.5. Число называется сильным псевдопростым числом Лукаса с параметрами P, Q ($slpsp(P, Q)$ если для $n - 1 = d2^s$, где d - нечетное, выполняется

$$\begin{aligned} &U_d = 0(mod\ n))\ or \\ &V_{d2^r} = 0(mod\ n)), \exists r : 0 \leq r < s \end{aligned} \tag{6}$$

Последовательности Лукаса обладают большим количеством свойств и еще большим количеством формул для получения параметров/коэффициентов. В этой работе будут использоваться три из них. Это рекуррентные формулы для получения значений U_n и формула для получения V_n из U_n:

$$\begin{aligned} U_{2k} &= 2U_kU_{k+1} - PU_k^2 \\ U_{2k+1} &= U_{k+1}^2 - QU_k^2 \end{aligned} \tag{7}$$

$$V_k = 2U_{k+1} - PU_k \tag{8}$$

Они позволяют получить значения U_n, V_n за по двоичному представлению числа n, следовательно сложность этой операции составляет $O(log\ n)$.

Далее приведем несколько теорем (без доказательств), чтобы лучше понимать, что происходит:

Theorem 2.1. *Пусть D - целое число, $n = \prod p_i^{\alpha_i}$ - нечетное положительное целое число, такое что $(n, D) = 1$. Тогда число значений P по модулю n, для которых существует Q, такое что $P^2 - 4Q = D\ (mod\ n)$ и выполняется (3), равно*

$$\prod_i ((\sigma(n), \sigma(p_i)) - 1) \tag{9}$$

Corollary 2.1.1. *Каждое нечетное составное целое число n является $lpsp(P, Q)$ для как минимум трех пар P, Q с конкретными значениями P по модулю n.*

3 Описание алгоритма

Описание алгоритма взято из [1]. Алгоритм заключается в том, чтобы проверить, является ли число $slpsp(P, Q)$ со специально подобранными значениями P, Q. Для этого мы с помощью метода Джона Селфриджа подберем P, Q, D с некоторыми ограничениями, а затем проверим несколько основных свойств псевдопростых чисел Лукаса, чтобы убедиться в простоте числа.

3.1 Метода Джона Селфриджа

Нам надо выбрать P, Q для заданного числа, чтобы построить на них последовательность Лукаса и найти $U_{\sigma(n)}$. Для начала, D - не должен быть квадратом по модулю n. В таком случае $(\frac{D}{n} = 1)$, $U_k = \frac{Q^{n-1}-1}{Q-1}$, тест сводится к простому тесту на простоту. Для предотвращения этого принято брать такое D, что $(\frac{D}{n}) = -1$. Метод Джона Селфриджа заключается в фиксации значения $P = 1$, и переборе возможных значений $D = 5\ , -7,\ 9,\ -11,\ 13, ...$, пока не найдется $D : (\frac{D}{n}) = -1$. После этого вычислим $Q = \frac{1-D}{4}$.

Отсюда вытекает, что n не должен быть целым квадратом, иначе $(\frac{D}{n}) > -1$ будет выполняться для всех D и подбор параметров станет бесконечным.

3.2 Алгоритм

1. Проверить делимость на 2;
2. Проверить на полный квадрат целого числа;
3. *Опционально.* Проверить, что число делится на небольшие простые (скажем, меньше 1000);
4. Проверить число на $spsp(2)$;
5. Выбрать P, Q, D с помощью метода Джона Селфриджа;
6. Проверить на $slpsp(P, Q)$;
7. Проверить выполнение дополнительных условий.

4 Реализация

Все алгоритмы реализованы на языке Python с модулем gmpy2.

4.1 Основные алгоритмы

Листинг 1: Нахождение U_n, U_{n+1} по модулю с параметрами

```
def lucas_U_seq(P, Q, mod, n):
  f_even = lambda Uk, Uk1: ((2*Uk*Uk1 - P*Uk**2) % mod, (Uk1**2 - Q*Uk**2) % mod)
  f_odd  = lambda Uk, Uk1: ((Uk1**2 - Q*Uk**2) % mod, (P*Uk1**2 - 2*Q*Uk*Uk1) % mod)
  f_cond = lambda Uk, Uk1, bit: (f_odd(Uk, Uk1)) if bit & 1 else (f_even(Uk, Uk1))
  accumulator = (0, 1)
  U = list()
  U.append(accumulator)
  for i in binary(n):
    accumulator = f_cond(accumulator[0], accumulator[1], i)
    U.append(accumulator)
  return U
```

Листинг 2: Нахождение V_n по U_n, U_{n+1}

```
def lucas_Vn(P, Q, Un, Un1, mod):
  return ((-P)*Un + 2*Un+1) % mod
```

Листинг 3: Проверка на spsp(2)

```
def spsp2(n) -> bool:
  d = n - 1
  s = 0
  while d % 2 == 0:
    d //= 2
    s += 1
  pow_d = gmpy2.powmod(2, d, n)
  if pow_d == 1:
    return True
  for r in range(s):
    if pow_d * gmpy2.powmod(2, gmpy2.powmod(2, r, n), n) % n == 1:
      return True

  return False
```

Листинг 4: Нахождение параметров методом Джона Селфриджа

```
def guess_params(n):
  # select the first D = {5,  7 , 9,  11 , 13,  15 , ...} such that (D/n) == -1
  # note that if D is square then (D/n) == 1
  if squareness(n):
    raise ValueError("{} is square".format(n))

  P = 1
  D = 5
  sign = -1
  while jacobi(D, n) != -1 or (1 - D) // 4 == -1 or (1 - D) // 4 == 1:
    D = -D + sign * 2
    sign *= -1

  Q = (1 - D) // 4

  return (P, Q, D)
```

Листинг 5: Основной код алгоритма

```
def lucas_pseudoprime_test(n) -> bool:
  # step 1: check for divisibility of some primes
  if n == 1:
```

```
4       return True
5     if first_primes_check:
6       for prime in primes:
7         if n % prime == 0 and n != prime:
8           print("{} divides by {}".format(n, prime))
9           return False
10
11    # step 2: spsp(2) test
12    if spsp2(n):
13      print("{} is spsp(2)".format(n))
14      return False
15
16    # step 3: get params using John Selfridge method, it includes squareness check
17    # we guarantee that (D/n) = -1 and Q != {1, -1}
18    P, Q, D = guess_params(n)
19    U_list = lucas_U_seq(P, Q, n, n + 1)
20    # get Un+1 and Un+2
21    Un1 = U_list[-1][0]
22    Un2 = U_list[-1][1]
23    if Un1 != 0:
24      return False
25
26    # step 4: slpsp(P, Q)
27    d = n + 1
28    s = 0
29    while d % 2 == 0:
30      d //= 2
31      s += 1
32    r = 1
33    gcdd = GCD(U_list[-s][0], n)
34    if 1 < gcdd < n:
35      print("GCD(Ue, n) <= 1 or GCD(Ue, n) >= n")
36      return False
37    while r <= s:
38      Ud2r1 = U_list[-r][0]
39      Ud2r2 = U_list[-r][1]
40      Vd2e1 = lucas_Vn(P, Q, Ud2r1, Ud2r2, n)
41      if 1 < GCD(Vd2e1, n) < n:
42        print("GCD(Vd2e1, n) <= 1 or GCD(Vd2e1, n) >= n")
43        return False
44      r += 1
45
46
47    # step 5: check additional condition
48    g2qd = GCD(n, 2*Q*D)
49
50    if 1 < g2qd <= n:
51      return False
52
53    # get Vn+1
54    Vn1 = lucas_Vn(P, Q, Un1, Un2, n)
55    if Vn1 == 2*Q:
56      return False
57
58    # there n may be prime or slpsp
59
60    return True
```

4.2 Вспомогательные алгоритмы

Листинг 6: Проверка на полный квадрат

```
def squareness(n) -> bool:
  return gmpy2.is_square(n)
```

Листинг 7: Находжение НОК

```
def GCD(x, y) -> int:
   return gmpy2.gcd(x, y)
```

Листинг 8: Бинарное разложение n

```
def binary(n):
  binary = []
  while (n > 0):
    binary.append(n%2)
    n //= 2
  return binary[::-1]
```

Листинг 9: Символ Якоби

```
def jacobi(a, n) -> int:
    if n <= 0:
        raise ValueError("'n' must be a positive integer.")
    if n % 2 == 0:
        raise ValueError("'n' must be odd.")
    a %= n
    result = 1
    while a != 0:
        while a % 2 == 0:
            a /= 2
            n_mod_8 = n % 8
            if n_mod_8 in (3, 5):
                result = -result
        a, n = n, a
        if a % 4 == 3 and n % 4 == 3:
            result = -result
        a %= n
    if n == 1:
        return result
    else:
        return 0
```

5 Тесты

Для тестирования было использовано несколько выборок с различными числами: числа Лукаса для некоторых параметров, числа Кармайкла, псевдопростые числа Эйлера, числа Каталана и выборки простых и составных чисел разной величины. Для выборок были построены таблицы с временем выполнения и для некоторых графики.

6 Источники

1. Robert Baillie; Samuel S. Wagstaff, Jr. (October 1980). "Lucas Pseudoprimes". Mathematics of Computation;

2. David Bressoud; Stan Wagon (2000). A Course in Computational Number Theory. New York: Key College Publishing in cooperation with Springer;

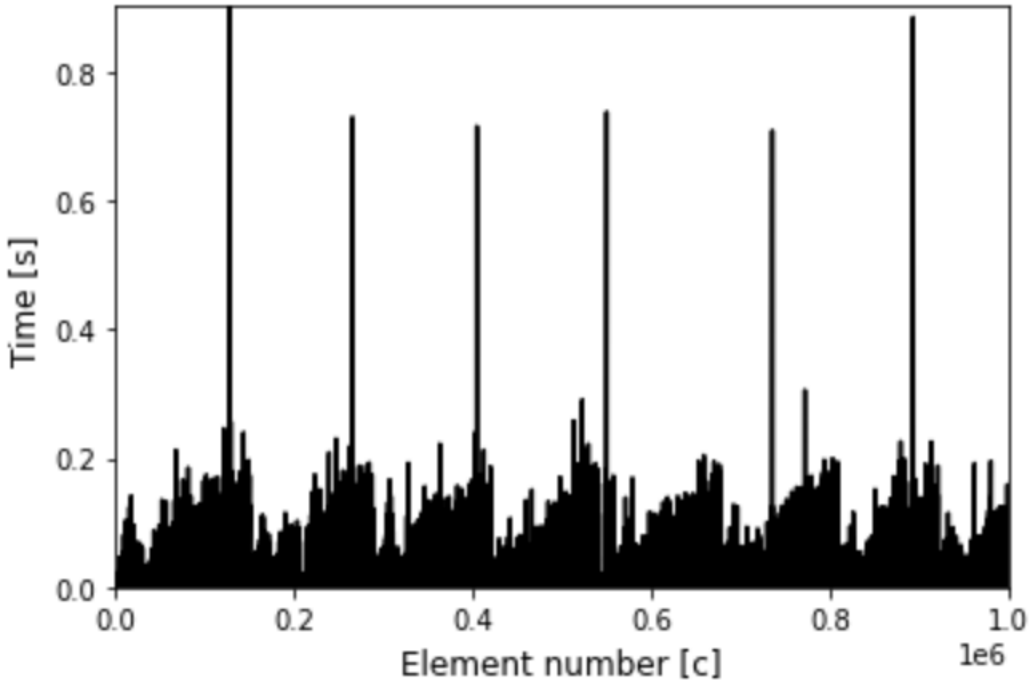

Рис. 1: Время работы на 100 000 простых чисел. Время в секундах

3. https://math.stackexchange.com/questions/1951853/bpsw-primality-test-selection-of-d-q-parameters;

4. https://en.wikipedia.org/wiki/Lucas_pseudoprime.

	numbers	time						
0	3	1307.249069						
1	5	2386.093140	**21**	647	35.285950	**41**	3889	61.035156
2	7	84.161758	**22**	743	30.040741	**42**	4423	52.690506
3	17	72.479248	**23**	839	29.563904	**43**	4759	32.424927
4	31	46.491623	**24**	941	54.597855	**44**	5507	62.942505
5	37	54.359436	**25**	1013	62.227249	**45**	5783	56.028366
6	43	56.743622	**26**	1069	52.213669	**46**	6133	75.340271
7	53	52.928925	**27**	1123	52.690506	**47**	6761	55.551529
8	59	50.306320	**28**	1231	39.339066	**48**	7001	67.949295
9	61	36.001205	**29**	1399	30.755997	**49**	7757	71.763992
10	67	46.253204	**30**	1483	64.611435	**50**	7901	56.266785
11	71	57.697296	**31**	1559	38.623810	**51**	8527	56.743622
12	89	36.478043	**32**	1607	31.948090	**52**	9007	32.663345
13	97	50.783157	**33**	1693	55.313110	**53**	9901	66.280365
14	109	45.299530	**34**	1733	56.266785	**54**	9949	80.823898
15	157	40.769577	**35**	1801	43.630600			
16	223	39.339066	**36**	1867	55.074692			
17	293	51.021576	**37**	2029	51.498413			
18	383	35.285950	**38**	2281	52.213669			
19	463	31.232834	**39**	2539	52.928925			
20	569	61.988831	**40**	3499	51.975250			

Рис. 2: Числа Простые и время их обработки (мкс)

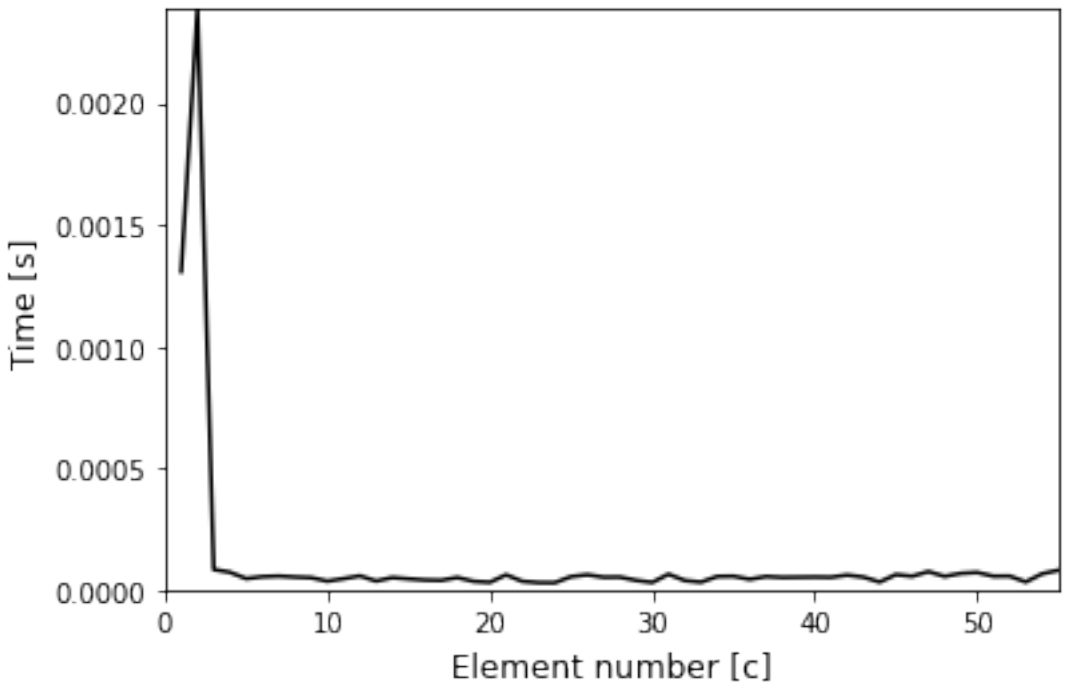

Рис. 3: Номера чисел Простых (см. таблицу выше) и время их обработки (сек)

	numbers	time
0	561	231.027603
1	41041	148.296356
2	825265	164.031982
3	321197185	234.603882
4	5394826801	213.623047
5	232250619601	267.267227
6	9746347772161	270.605087
7	1436697831295441	396.728516
8	60977817398996785	345.706940
9	7156857700403137441	475.168228
10	1791562810662585767521	502.824783
11	87674969936234821377601	494.718552
12	65531309267520060314817 61	695.705414
13	15902312310431783769516984 01	591.993332
14	35237869211718889547310642241	555.038452

Рис. 4: Числа Кармайкла и время их обработки (мкс)

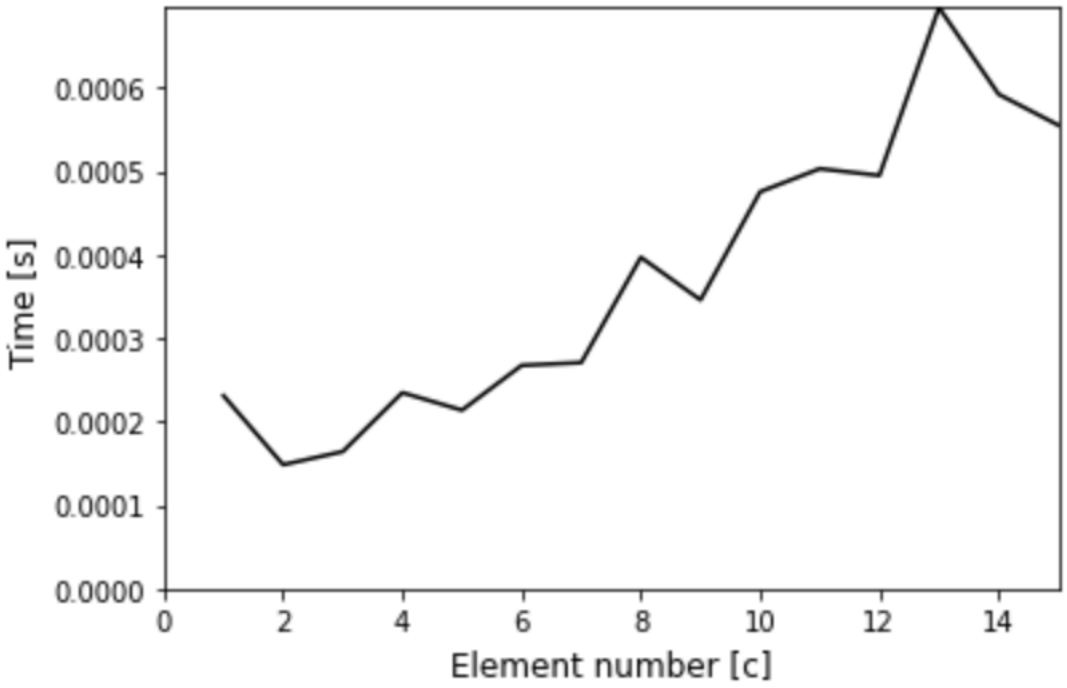

Рис. 5: Номера чисел Кармайкла (см. таблицу выше) и время их обработки (сек)

	numbers	time
0	561	82.969666
1	1105	35.047531
2	1729	39.100647
3	1905	35.762787
4	2047	73.194504
5	2465	57.935715
6	3277	60.796738
7	4033	72.479248
8	4681	39.815903
9	6601	66.757202
10	8321	42.438507
11	8481	41.007996
12	10585	50.306320

Рис. 6: Числа Эйлера и время их обработки (мкс)

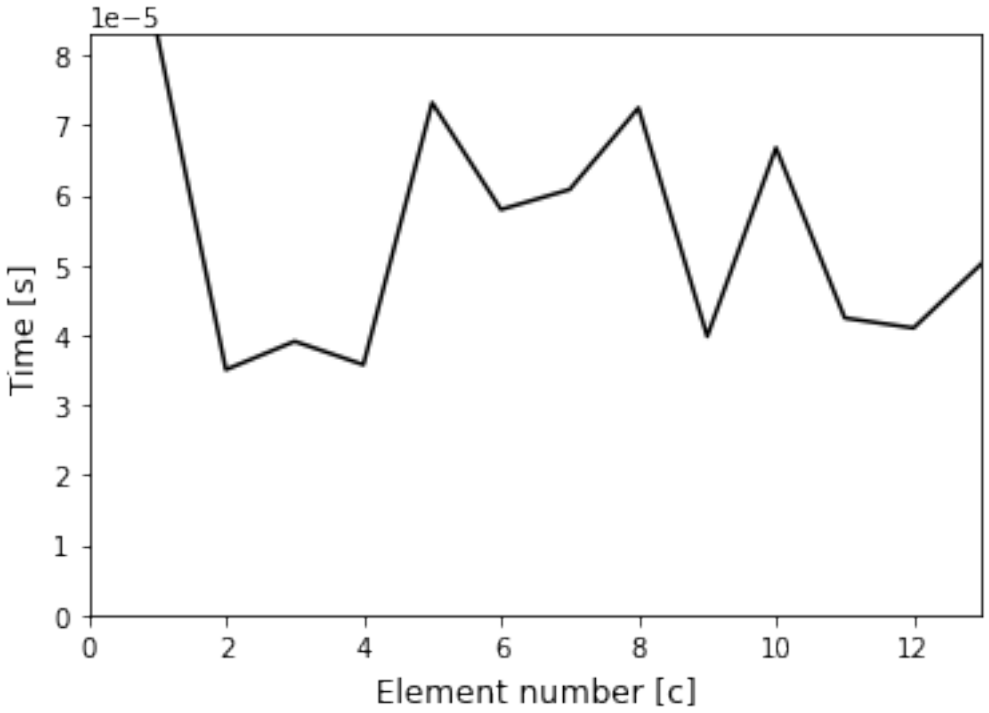

Рис. 7: Номера чисел Эйлера (см. таблицу выше) и время их обработки (сек)

	numbers	time
0	1	3.814697
1	7	2205.133438
2	16	45.061111
3	43	78.678131
4	105	62.465668
5	255	82.254410
6	646	15.020370
7	1547	61.511993
8	3605	64.373016
9	8241	54.836273
10	19279	50.067902
11	44706	6.437302
12	105212	3.576279
13	246683	55.789948

Рис. 8: Числа Майка и время их обработки (мкс)

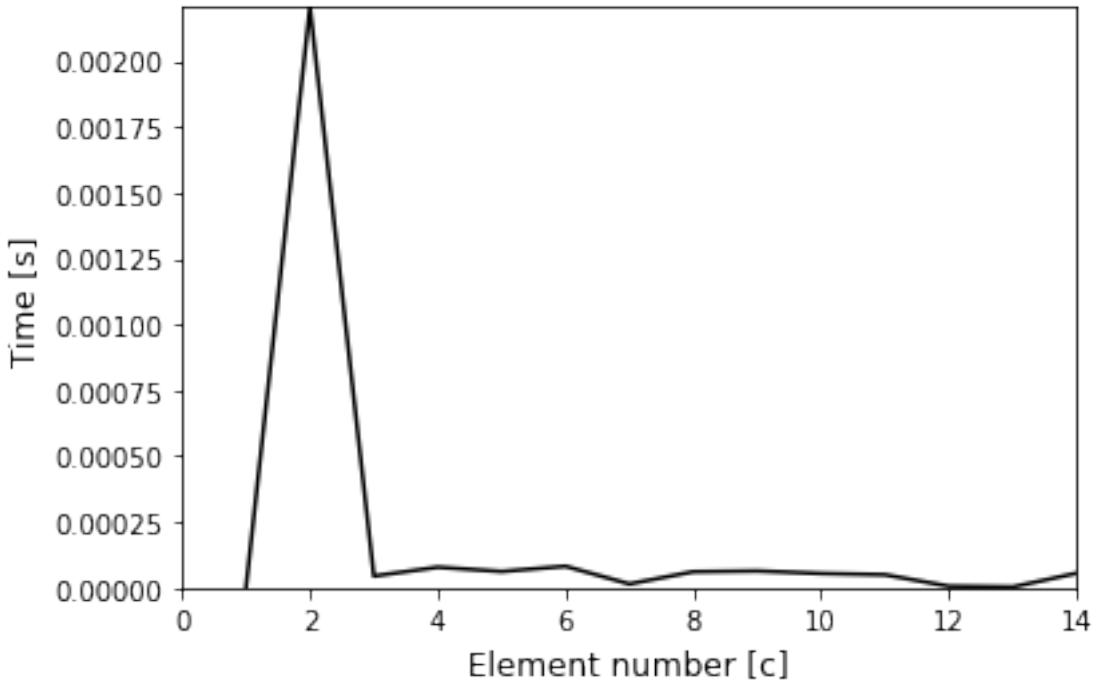

Рис. 9: Номера чисел Майка (см. таблицу выше) и время их обработки (сек)