მარტივი რიცხვების ძებნის ალგორითმებისა და მათი გაუმჯობესებული ვერსიების რეალიზება Python დაპროგრამების ენაზე

ლელა გაჩეჩილაძე, ნანა კურკუმული, ლია ნონიკაშვილი
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

წარმოდგენილია ნატურალურ რიცხვთა მიმდევრობიდან მარტივი რიცხვების ძებნის პროგრამული რეალიზაციები Python დაპროგრამების ენაზე. ჩვენ მიერ შემუშავებულ პროგრამებში გამოყენებულია მარტივი რიცხვების პოვნის როგორც საყოველთაოდ ცნობილი ერატოსთენეს „ცხავის“ ალგორითმი, ასევე სუნდარამის „ცხავის“ ნაკლებად ცნობილი, მაგრამ საკმაოდ დახვეწილი ალგორითმი და აღნიშნული ალგორითმების ჩვენ მიერ შემუშავებული გაუმჯობესებული ვერსიები. განხილულია ალგორითმების სირთულის საკითხები და მათი ანალიზის საფუძველზე გაკეთებულია შესაბამისი დასკვნები.

საკვანძო სიტყვები: მარტივი და შედგენილი რიცხვები. დეტერმინირებული ალგორითმები. ალგორითმების სირთულე. სიმრავლეები.

1. შესავალი

მათემატიკის კურსიდან ცნობილია, რომ მარტივი რიცხვი ეწოდება ნატურალურ რიცხვს, რომელსაც მხოლოდ ორი განსხვავებული ნატურალური გამყოფი აქვს: 1 და თავისი თავი [1].

ელინიზმის ეპოქის უდიდესმა მოაზროვნემ, მათემატიკოსმა ევკლიდე ალექსანდრიელმა ძვ.წ. დაახლოებით 300 წ. აჩვენა, რომ მარტივი რიცხვების რაოდენობა უსასრულოა, თუმცა მარტივების სიმკვრივე ნატურალურ რიცხვებში 0-ის ტოლია [2]. განსაზღვრების თანახმად, 1 არ არის მარტივი რიცხვი. არითმეტიკის ფუნდამენტური თეორემა წარმოგვიდგენს მარტივი რიცხვების მნიშვნელობას რიცხვთა თეორიაში: ნებისმიერი ერთზე მეტი ნატურალური რიცხვი შეიძლება დაიშალოს მარტივი რიცხვების უნიკალურ ნამრავლად.

n რიცხვის მარტივობის შესამოწმებელი უმარტივესი ალგორითმი შემდეგია:

შემოწმდეს, იყოფა თუ არა *n* რიცხვი ნებისმიერ მთელ რიცხვზე 2-დან $\sqrt{n}$-ის ჩათვლით. თუ *n* იყოფა ამ შუალედში მოთავსებულ რომელიმე მთელ რიცხვზე, მაშინ ის მარტივ რიცხვს არ წარმოადგენს და მას შედგენილი რიცხვი ეწოდება. ხოლო თუ *n* ამ შუალედში მოთავსებულ არც ერთ მთელ რიცხვზე არ იყოფა, მაშინ ის მარტივია. ამ ალგორითმს $\sqrt{n}$ რაოდენობის გაყოფის ოპერაციის ჩატარება ესაჭიროება, რაც, ცხადია, დიდი რიცხვებისთვის მეტად მოუხერხებელია.

მარტივი რიცხვების თეორმის თანახმად, შემთხვევითად არჩეული *n* რიცხვის მარტივობის ალბათობა *n* რიცხვის ციფრების რაოდენობის ან მისი ლოგარითმის უკუპროპორციულია.

მიუხედავად მარტივი რიცხვების ფართომასშტაბიანი შესწავლისა, მათემატიკაში დღემდე არსებობს მათთან დაკავშირებული გადაუჭრელი ამოცანები.

მარტივი რიცხვები ფართოდ გამოიყენება ინფორმატიკასა და კრიპტოგრაფიაში (რამეთუ დიდი რიცხვების მარტივ მამრავლებად დაშლა რთული პროცედურაა). დიდი მარტივი რიცხვების ძიებითაა მოტივირებული განსაკუთრებული მარტივი რიცხვების შესწავლა: მაგალითად, მერსენის მარტივი რიცხვები, რომელთა მარტივობა შედარებით ადვილი შესამოწმებელია [3]. 2010 წლის მონაცემებით, ჩვენთვის ცნობილი უდიდესი მარტივი რიცხვის ათობით ჩანაწერი დაახლოებით 13 მილიონი ციფრისგან შედგება.

2. ძირითადი ნაწილი

დღეს რიცხვის მარტივობის შესამოწმებლად უფრო დახვეწილი ალგორითმები არსებობს. წინამდებარე სტატიაში ამ მიზნით ჩვენ ერატოსთენესა და სუნდარამის ალგორითმებს, მათ გაუმჯობესებულ ვერსიებსა და პროგრამულ რეალიზაციებს წარმოგიდგენთ Python დაპროგრამების ენაზე.

ძველი ბერძენი მათემატიკოსის ერატოსთენეს „ცხავის“ ალგორითმი შემდეგი ბიჯების შესრულებას ითვალისწინებს [5]:

1) მიმდევრობით დავწეროთ ყველა მთელი რიცხვი 2-დან n-მდე (2, 3, 4, ..., n);

2) დავუშვათ, რომ p ცვლადის საწყისი მნიშვნელობა 2-ის ტოლია (პირველი მარტივი რიცხვის);

3) მიმდევრობაში $2p$-დან n-მდე გადავხაზოთ ყველა რიცხვი, რომელიც p-ზე უნაშთოდ იყოფა (ანუ, რიცხვები: $2p$, $3p$, $4p$, ...);

4) მოვძებნოთ პირველი p-ზე მეტი მნიშვნელობის მქონე გადაუხაზავი რიცხვი და მისი მნიშვნელობა მივანიჭოთ p ცვლადს;

5) მე-3 და მე-4 ბიჯები გავიმეოროთ, სანამ p არ აღმოჩნდება n-ზე მეტი;

6) მიმდევრობაში ყველა გადაუხაზავი რიცხვი - მარტივი რიცხვია.

ერატოსთენეს „ცხავის“ ალგორითმის შესაბამის პროგრამულ რეალიზაციას Python დაპროგრამების ენაზე ქვემოთ ნაჩვენები სახე აქვს:

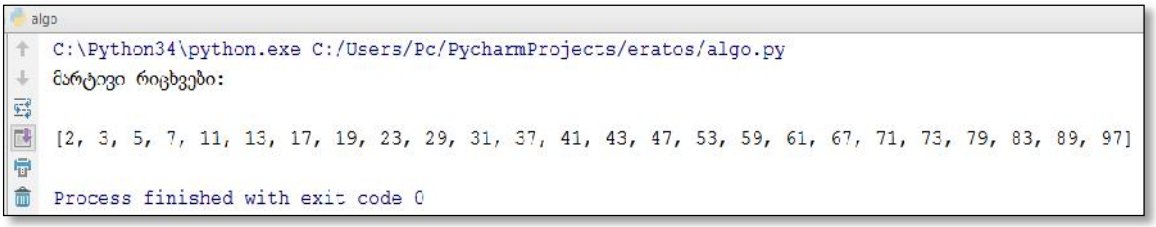
```
#ერატოსთენეს ალგორითმის პროგრამული რეალიზაცია
def primes(n):
    a = [0] * n  #n რაოდენობის ელემენტებისგან მასივის შექმნა
    for i in range(n):  # მასივის შევსება
        a[i] = i  #0-დან  n-1 - ის ჩათვლით მნიშვნელობებით

    # მეორე ელემენტის მნიშვნელობა ერთის ტოლია,
    # რაც მარტივ რიცხვად არ ითვლება
    # ვანიჭებთ მას 0-ის ტოლ მნიშვნელობას
    a[1] = 0
    m = 2
    while m < n:  # ყველა ელემენტის შემოწმება
        if a[m] != 0:  # თუ ელემენტის მნიშვნელობა ნულისგან
                       # განსხვავებულია, ვზრდით ორჯერ
            j = m * 2  # (მიმდინარე ელემენტი მარტივი რიცხვია)
            while j < n:
                a[j] = 0
                j = j + m
        m += 1
```

```
    # მარტივი რიცხვების კონსოლზე გამოტანა
    b = []
    for i in a:
        if a[i] != 0:
            b.append(a[i])

    del a
    return b
print("მარტივი რიცხვები:")
print()
print(primes(100))
```

პროგრამის შესრულების შედეგს შემდეგი სახე აქვს:



წარმოდგენილი ალგორითმი შეგვიძლია გავაუმჯობესოთ. კერძოდ, მე-3 ბიჯზე პირდაპირ p^2 რიცხვიდან დაწყებული რიცხვები შეგვიძლია გადავხაზოთ, რადგან ყველა მასზე ნაკლები შედგენილი რიცხვი ამ დროისათვის უკვე გადახაზული იქნება და შესაბამისად, ალგორითმის დასრულება შესაძლებელი გახდება, როდესაც შესრულდება პირობა: $p^2 > n$.

საჭიროა აღვნიშნოთ, რომ ალგორითმის სირთულე შეადგენს $O(n\log\log n)$ ან $O(n(\log n)(\log\log n))$ ბიტურ ოპერაციებს კომპიუტერის ოპერატიული მეხსიერების (RAM) გამოთვლების მოდელში.

ალგორითმის სირთულე ინფორმატიკასა და ალგორითმების თეორიაში ცნობილი ტერმინია. ეს უკანასკნელი განსაზღვრავს სამუშაოს მოცულობას, რომელიც გარკვეული ალგორითმის მიერ სრულდება საწყისი მონაცემების ზომიდან გამომდინარე. სამუშაოს მოცულობა იზომება დროისა და სივრცის აბსტრაქტული გაგებით, რომელთაც გამოთვლითი რესურსები ეწოდება. დრო განისაზღვრება ამოცანის გადასაწყვეტად საჭირო აუცილებელი ელემენტარული ბიჯების რაოდენობით, მაშინ, როდესაც სივრცეს კომპიუტერის ოპერატიული მეხსიერების მოცულობა განაპირობებს. ამ შემთხვევაში ალგორითმის შემუშავების ცენტრალურ კითხვაზე ვცდილობთ პასუხის გაცემას. კერძოდ, როგორ შეიცვლება ალგორითმის შესრულების დრო და დაკავებული მეხსიერების მოცულობა საწყისი მონაცემების ზომაზე დამოკიდებულებით?

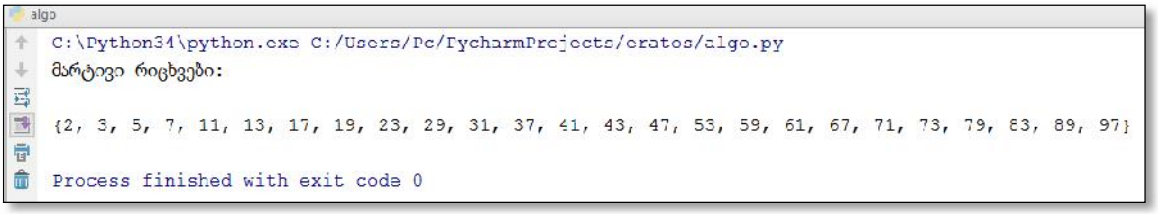
ერატოსთენეს „ცხავი“-ს ალგორითმის ჩვენ მიერ მოდიფიცირებული ვარიანტი მონაცემთა სტრუქტურის ისეთი ტიპის გამოყენებას ემყარება, როგორიცაა სიმრავლე.

ამ უკანასკნელის პროგრამულ რეალიზაციას ქვემოთ წარმოდგენილი სახე აქვს:

```
import math
#ერატოსთენეს ალგორითმის მოდიფიკაცია
def primes(N):
```

```
  sieve = set(range(2, N))
  for i in range(2, int(math.sqrt(N))):
    if i in sieve:
      sieve -= set(range(2*i, N, i))
  return sieve
print("მარტივი რიცხვები:")
print()
print (primes(100))
```

პროგრამის შესრულების შედეგს შემდეგი სახე აქვს:



ახლა კი სუნდარამის „ცხავის“ ალგორითმი განვიხილოთ, რომელიც მათემატიკაში დეტერმინირებულ ალგორითმს წარმოადგენს (ანუ ალგორითმულ პროცესს, რომელიც უნიკალურ და წინასწარ განსაზღვრულ შედეგებს გვაძლევს მოცემული საწყისი მნიშვნელობების დროს). ეს უკანასკნელი ინდოელი სტუდენტის ს.პ. სუნდარამის მიერ იქნა შემუშავებული [4].

ალგორითმი ერთისაგან განსხვავებული კენტი მნიშვნელობის ნატურალურ რიცხვებზე მუშაობს, რომლებიც $2m+1$ სახითაა წარმოდგენილი, სადაც m ნატურალური რიცხვია. თუ $2m+1$ რიცხვი შედგენილია, მაშინ ის ერთისაგან განსხვავებული ორი კენტი რიცხვის ნამრავლის სახით წარმოდგება:

$$2m + 1 = (2i + 1)(2j + 1),$$

სადაც i და j ნატურალური რიცხვებია. აქ მართებულია შემდეგი ტოლობაც:

$$m = 2ij + i + j.$$

ამგვარად, თუ ნატურალური რიცხვების სიიდან გამოვრიცხავთ ყველა $2ij + i + j$ სახის რიცხვს, სადაც $1 \le i \le j$, მაშინ ყოველი დარჩენილი m რიცხვისათვის $2m + 1$ რიცხვი „ვალდებულია“ იყოს მარტივი. და, პირიქით, თუ $2m + 1$ რიცხვი მარტივია, მაშინ m რიცხვის წარმოდგენა $2ij + i + j$ სახით შეუძლებელია და შესაბამისად, m რიცხვი ალგორითმის მუშაობის პროცესში არ გამოირიცხება.

ამგვარად, სუნდარამის „ცხავის“ ალგორითმი შემდეგი ბიჯების შესრულებას ითვალისწინებს:

1) [1;N) ნატურალური რიცხვების მიმდევრობიდან საჭიროა გამოირიცხოს ყველა $2ij + i + j$ რიცხვი, სადაც $i \le j$ ინდექსები ყველა ნატურალურ მნიშვნელობას გადის და რომელთათვისაც სამართლიანია უტოლობა: $i + j + 2ij \le N$. ამასთან, მნიშვნელობები განისაზღვრება შემდეგი ფორმულებით:

2) $i = 1, 2, \dots, \left[\frac{\sqrt{2N+1}-1}{2}\right]$ და $j = i, i + 1, \dots, \left[\frac{N-i}{2i+1}\right]$.

3) ყოველი დარჩენილი რიცხვი უნდა გავამრავლოთ 2-ზე და ერთით გავზარდოთ;

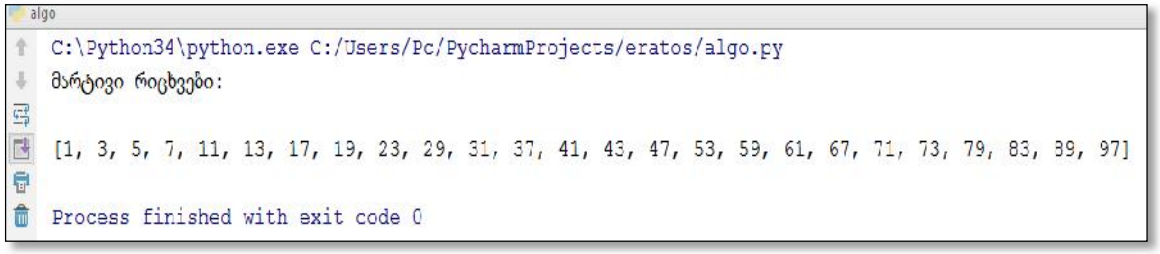
4) მიღებული მიმდევრობა წარმოადგენს ყველა კენტი მნიშვნელობის მარტივ რიცხვს [1, 2N+1] მონაკვეთზე.

ყურადსაღებია ის ფაქტი, რომ ალგორითმის შემავალი პარამეტრია N, ხოლო მარტივი რიცხვები განისაზღვრება 2N+1-ის ჩათვლთ.

სუნდარამის „ცხავის“ ალგორითმის შესაბამის პროგრამულ რეალიზაციას Python დაპროგრამების ენაზე ქვემოთ ნაჩვენები სახე აქვს:

```
#სუნდარამის ალგორითმის პროგრამული რეალიზაცია:
def sundaram(N):
    D=N//2
    B=N//6
    A=set(range(D))
    for i in range(1,B+1):
        for j in range(i,(D+i)//(1+2*i)+1):
            A.discard(i+j+2*i*j)
    A=[ 2*x+1 for x in A ]
    return A
print("მარტივი რიცხვები:")
print()
print(sundaram(100))
```

პროგრამის შესრულების შედეგს შემდეგი სახე აქვს:



3. დასკვნა

წარმოდგენილი პროგრამული რეალიზაციებიდან გამომდინარე, ჩვენ შემდეგი შედეგები მივიღეთ: [2;650000] ინტერვალიდან მარტივი რიცხვების განსაზღვრის დროს უკეთეს შედეგს ის რეალიზაცია გვაძლევს, რომელიც სიმრავლის გამოყენებას ეყრდნობა. აღნიშნული ინტერვალის ზემოთ კი უპირატესობა ერატოსთენეს საწყის ალგორითმს ენიჭება, რაც გარკვეულწილად Python-ენაში სიმრავლეების შიდა რეალიზაციებს უკავშირდება.

ლიტერატურა - References – Литература:

1. Crandall R. (2000). Prime Numbers. A Computational Perspective. Second Edition. Portland. USA

2. Heath Th.L. (2012). Euclide. The Thirteen Books of The Elements. Kindle Edition

3. Silverman J.H. (2017). A Friendly Introduction to Number Theory. Fourth Edition. Pearson

4. Aiyar V. Ramaswami. (2004). Sundaram's Sieve for Prime Numbers. eJournal/eMagazine Indian Mathematical Society. The Mathematics Student. **2** (2): 73.

5. გაჩეჩილაძე ლ. დაპროგრამების ენა Python. საქართველოს ტექნიკური უნივერსიტეტი. 2018.

IMPLEMENTATION OF ALGORITHMS FOR FINDING PRIME NUMBERS AND THEIR IMPROVED VERSIONS ON THE PYTHON PROGRAMMING LANGUAGE

Gachechiladze Lela, Kurkumuli Nana, Nonikashvili Lia
Georgian Technical University

Summary

The paper represented the software implementations of finding primes from a sequence of natural numbers on the Python programming language. In the programs developed by us, they are used as the generally recognized “Sieve” algorithm of Eratosthenes, also the lesser known, but rather sophisticated “Sieve” algorithm by Sundaram and the improved versions of the indicated algorithms for finding prime numbers developed by us. The problems of the algorithms complexity are considered and relevant conclusions are made based on their analysis.

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОИСКА ПРОСТЫХ ЧИСЕЛ И ИХ УЛУЧШЕННЫХ ВЕРСИЙ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Гачечиладзе Л., Куркумули Н., Ноникашвили Л.
Грузинский Технический Университет

Резюме

Представлены программные реализации поиска простых чисел из последовательности натуральных чисел на языке программирования Python. В разработанных нами программах использованы как общепризнанный алгоритм “Решето” Эратосфена, также менее известный, но довольно утонченный алгоритм “Решето” Сундарама и разработанные нами улучшенные версии указанных алгоритмов нахождения простых чисел. Рассмотрены вопросы сложности алгоритмов и на основе их анализа сделаны соответствующие выводы.