# Σύγκριση αλγορίθμων προσδιορισμού ενός πρώτου αριθμού

Μανιατάκος Παναγιώτης

Εαρινό εξάμηνο 2023

## 1 Πρώτοι αριθμοί- Εισαγωγή στο πρόβλημα

Στα μαθηματικά πρώτος αριθμός (ή απλά πρώτος) είναι ένας φυσικός αριθμός με την ιδιότητα οι μόνοι φυσικοί διαιρέτες του να είναι η μονάδα και ο εαυτός του.
Υπάρχουν άπειροι σε πλήθος πρώτοι αριθμοί, όπως απέδειξε ο Ευκλείδης περίπου στο 300 π.Χ., ενώ δεν υπάρχει κανένας γνωστός τύπος ο οποίος να διαχωρίζει όλους τους πρώτους αριθμούς από τους σύνθετους.
Οι πρώτοι αριθμοί είναι ένα από τα αντικείμενα της θεωρίας αριθμών και είναι μια πολύ ενεργή ερευνητικά περιοχή των μαθηματικών. Πολλά ερωτήματα γύρω από τους πρώτους αριθμούς παραμένουν ανοιχτά, όπως η υπόθεση Ρίμαν, η εικασία του Γκόλντμπαχ, η οποία λέει ότι κάθε άρτιος ακέραιος μεγαλύτερος του 2 μπορεί να γραφεί ως άθροισμα δύο πρώτων και η εικασία των διδύμων πρώτων, η οποία λέει ότι υπάρχουν άπειρα σε πλήθος ζευγάρια πρώτων των οποίων η διαφορά είναι 2. Τέτοιες ερωτήσεις οδήγησαν στην ανάπτυξη διάφορων κλάδων της θεωρίας αριθμών, εστιάζοντας στην αναλυτική ή αλγεβρική πλευρά των αριθμών. Οι πρώτοι χρησιμοποιούνται σε πολλούς τομείς στην τεχνολογία πληροφοριών, όπως στην Κρυπτογράφηση Δημόσιου Κλειδιού, η οποία χρησιμοποιεί ιδιότητες, όπως τη δυσκολία να αναλύεις ένα μεγάλο αριθμό σε γινόμενο πρώτων αριθμών.
Έτσι λοιπόν εύκολα μπορεί να καταλάβει κανείς πως ο έλεγχος σχετικά με το αν ένας φυσικός αριθμός είναι πρώτος αποτελεί ένα βασικό μαθηματικό πρόβλημα

## 2 Bruteforce Algorithm

### 2.1 Περιγραφή της μεθόδου

Στην πρώτη μέθοδο ($Bruteforce$) εξετάζονται διαδοχικά όλοι οι αριθμοί $M \leq N$, που $N$ ο αριθμός ο οποίος εξετάζουμε.Χρησιμοποιούμε εναν μετρητή για τους διαιρετες του αριθμού ο οποίος ξεκινάει απο το 1 (αφού το 1 ειναι πάντα διαιρέτης).Μόλις βρεθεί διαιρέτης του $N$ ο μετρητής αυξάνεται κατα 1 και αν εξαντληθούν οι $M$ χωρίς να βρεθεί διαιρέτης,πέρα απ'ο τον ίδιο τον αριθμό, τότε ο $N$ είναι πρώτος.

## 2.2 Ο αλγόριθμος σε ψευδογλώσσα

**Script BruteForce**
Εκτύπωσε 'Δώσε ακέραιο αριθμό n='
Read $n$
@time begin
$counter = 1$
for $i = 2 : n$
if $(rem(n, i)) == 0$
$counter = counter + 1$
end
end
if $counter == 2$
Εκτύπωσε 'Ο $n$ είναι πρώτος'
else
Εκτύπωσε 'Ο $n$ δεν είναι πρώτος'
end
@time end

## 2.3 Ο αλγόριθμος σε Matlab

```
% primes1.m
n=0;
while n<2
 n = input('Give intenger greater than 1: ');
end
counter = 1;
for i=2:n
 if (rem(n,i)==0)
 counter = counter+1;
 end
end
if (counter==2)
fprintf('%d is prime \n', n);
else
 fprintf('%d is not prime\n', n);
end
```

## 2.4 Ο αλγόριθμος σε Julia

```
function primes(n::Number)
 # primes1.m
 counter= 1
 for i in 2:n
```

```
 5     if (rem(n,i)==0)
 6       counter=counter+1
 7     end
 8    end
 9    if counter==2
10     print("$n is prime.")
11    else
12     print("$n is not prime.")
13    end
14   end
```

## 2.5 Ο αλγόριθμος σε Python

```
 1  import time
 2  import math
 3  # primes1.m
 4  n= int(input("Give intenger greater than 1: "))
 5  st = time.time()
 6  counter= 1
 7  for i in range(1,n+1):
 8    if (n%i)==0:
 9     counter=counter+1
10  if counter==1:
11   print( n ," is prime .")
12  else:
13   print(n " ,is not prime.")
14  et = time.time()
15  elapsed_time = et - st
16  print('Execution time:', elapsed_time, 'seconds')
```

## 2.6 Αριθμητική υλοποίηση

Πολυπλοκότητα: O(n)

Οι επιδόσεις των τριών αλγορίθμων:

| n | Octave | Matlab | Julia | Python |
|---|---|---|---|---|
| 1,000,000 | 4.33 sec | $2.5 \cdot 10^{-2}$ sec | $4 \cdot 10^{-3}$ sec | 0.15 sec |
| 10,000,000 | 42 sec | 0.15 sec | 0.03 sec | 1 sec |
| 100,000,000 | 7 min 35 sec | 1.41 sec | 0.35 sec | 10.06 sec |
| 1,000,000,000 | 1 hour and 11 min | 15.03 sec | 3.69 sec | 99.82 sec |

All of the programs were executed in a pc with:
an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz Processor

16 GB of RAM memory
an NVIDIA GeForce GTX 1660 graphics card
and an a Samsung EVO 870 SSD hard drive

# 3 Improved Bruteforce

## 3.1 Περιγραφή της μεθόδου

Τώρα βασιζόμενοι στην παρατήρηση ότι κανένας αριθμός $N$ δεν έχει διαιρέτη μεγαλύτερο του $N/2$, τροποποιούμε τον παραπάνω αλγόριθμο εξετάζοντας όλους τους αριθμούς $M \leq N/2$, διπλασιάζοντας έτσι την ταχύτητα σε σχέση με την πρώτη μέθοδο.Στην περίπτωση αυτή για να είναι πρώτος ο μετρητής θα πρέπει να έχει την τιμή 1.

## 3.2 Ο αλγόριθμος σε ψευδογλώσσα

**Script Improved**
Εκτύπωσε 'Δώσε ακέραιο αριθμό n='
Read $n$
@time begin
$counter = 1$
b =$(n/2)$
for $i = 2 : b$
if $(rem(n, i) == 0$
$counter = counter + 1$
end
if $counter == 1$
Εκτύπωσε 'Ο $n$ είναι πρώτος'
else
Εκτύπωσε 'Ο $n$ δεν είναι πρώτος'
end
@time end

## 3.3 Ο αλγόριθμος σε Matlab

```
1  % primes2.m
2  n=0;
3  while n<2
4   n = input('Give intenger greater than 1: ');
5  end
6  counter = 1;
```

```
 7  b= n/2
 8  for i=2:(b)
 9   if (rem(n,i)==0)
10    counter = counter+1;
11   end
12  end
13  if (counter==1)
14  fprintf('%d is prime \n', n);
15  else
16   fprintf('%d is not prime\n', n);
17  end
```

### 3.4 Ο αλγόριθμος σε Julia

```
 1  function primes(n::Number)
 2   # primes2.m
 3   counter= 1
 4   for i in 2:fld(n,2)
 5    if (rem(n,i)==0)
 6      counter=counter+1
 7    end
 8   end
 9   if counter==1
10    print("$n is prime.")
11   else
12    print("$n is not prime.")
13   end
14  end
```

### 3.5 Ο αλγόριθμος σε Python

```
 1  import time
 2  import math
 3  # primes1.m
 4  n= int(input("Give intenger greater than 1: "))
 5  st = time.time()
 6  counter= 1
 7  b = math . floor ((n +1) /2)
 8  for i in range(1,b):
 9    if (n%i)==0:
10     counter=counter+1
11  if counter==1:
12   print( n ," is prime .")
13  else:
```

```
14    print(n ," is not prime.")
15   et = time.time()
16   elapsed_time = et - st
17   print('Execution time:', elapsed_time, 'seconds')
```

### 3.6 Αριθμητική υλοποίηση

Πολυπλοκότητα: O(n/2)

Οι επιδόσεις των τριών αλγορίθμων:

| n | Octave | Matlab | Julia | Python |
|---|---|---|---|---|
| 1,000,000 | 2.12 sec | $9 \cdot 10^{-3}$ sec | $2.4 \cdot 10^{-2}$ sec | 0.06 sec |
| 10.000.000 | 21.24 sec | $7.1 \cdot 10^{-2}$ sec | $2.1 \cdot 10^{-2}$ sec | 0.47 sec |
| 100,000,000 | 3 min and 32 sec | 0.70 sec | 0.20 sec | 4,77 sec |
| 1,000,000,000 | 37min and 15 sec | 6.82 sec | 2.09 sec | 47,17 sec |

All of the programs were executed in a pc with: an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz Processor
16 GB of RAM memory
an NVIDIA GeForce GTX 1660 graphics card
and an a Samsung EVO 870 SSD hard drive

## 4 The root Algorithm

### 4.1 Περιγραφή της μεθόδου

Στην μέθοδο αυτή παρατηρούμε ότι αν ένας αριθμός $N$ δεν είναι πρώτος τότε έχει (τουλάχιστον) δύο διαιρέτες μεγαλύτερους από 1. Σε αυτήν την περίπτωση τουλάχιστον ένας διαιρέτης είναι μικρότερος από την τετραγωνική ρίζα του αριθμού. Τροποποιούμε την δεύτερη μέθοδο εξετάζοντας όλους τους αριθμούς $M$ που είναι μικρότεροι από την τετραγωνική ρίζα του $N$, αν η τελευταία δεν είναι ακέραιος. Αλλιώς ο αριθμός δεν είναι πρώτος, επειδή τον διαιρεί και η τετραγωνική του ρίζα.

### 4.2 Ο αλγόριθμος σε ψευδογλώσσα

**script RootAlgorithm**
Εκτύπωσε 'Δώσε ακέραιο αριθμό n='
Read $n$
@time begin
$counter = 1$
b = $floor(sqrt(n)) + 1$

for $i = 2 : b$
if $(rem(n, i) == 0$
$counter = counter + 1$
end
if $counter == 1$
Εκτύπωσε 'Ο $n$ είναι πρώτος'
else
Εκτύπωσε 'Ο $n$ δεν είναι πρώτος'
end
@time end

## 4.3 Ο αλγόριθμος σε Matlab

```
% primes3.m
n=0;
while n<2
 n = input('Give intenger greater than 1: ');
end
counter = 1;
b= floor(sqrt(n))+ 1
for i=2: b
 if (rem(n,i)==0)
 counter = counter+1;
 end
end
if (counter==1)
fprintf('%d is prime \n', n);
else
 fprintf('%d is not prime\n', n);
end
```

## 4.4 Ο αλγόριθμος σε Julia

```
function primes(n::Number)
 # primes3.m
 counter= 1
 b= sqrt(fld(n,2))+ 1
 for i in 2:b
  if (rem(n,i)==0)
    counter=counter+1
  end
 end
 if counter==1
```

```
    print("$n is prime.")
   else
    print("$n is not prime.")
   end
  end
```

## 4.5 Ο αλγόριθμος σε Python

```
import time
import math
# primes1.m
n= int(input("Give intenger greater than 1: "))
st = time.time()
counter= 1
b= int(np.sqrt((n)))+ 1
for i in range(1,b):
  if (n%i)==0:
    counter=counter+1
if counter==1:
 print( n " is prime .")
else:
 print(n " is not prime.")
et = time.time()
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')
```

## 4.6 Αριθμητική υλοποίηση

Πολυπλοκότητα: $O(\sqrt{n})$

Οι επιδόσεις των τριών αλγορίθμων:

| n | Octave | Matlab | Julia | Python |
|---|---|---|---|---|
| 1,000,000 | $4.8 \cdot 10^{-3}$ sec | $2.5 \cdot 10^{-4}$ sec | $1,7 \cdot 10^{-4}$ sec | $1 \cdot 10^{-2}$ sec |
| 10.000.000 | $1.4 \cdot 10^{-2}$ sec | $4.4 \cdot 10^{-4}$ sec | $2,2 \cdot 10^{-4}$ sec | $1.1 \cdot 10^{-2}$ sec |
| 100,000,000 | $4.4 \cdot 10^{-2}$ sec | $8.3 \cdot 10^{-4}$ sec | $3.1 \cdot 10^{-4}$ sec | $1.4 \cdot 10^{-2}$ sec |
| 1,000,000,000 | 0.13 sec | $2.3 \cdot 10^{-3}$ sec | $6.6 \cdot 10^{-4}$ sec | $1.6 \cdot 10^{-2}$ sec |

All of the programs were executed in a pc with:
an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz Processor
16 GB of RAM memory
an NVIDIA GeForce GTX 1660 graphics card
and an a Samsung EVO 870 SSD hard drive

# 5 Συμπεράσματα

Όσον αφορά τους αλγορίθμους καθαυτούς, παρατηρούμε ότι η μέθοδος της ρίζας (Πολυπλοκότητα: $O(\sqrt{n})$) είναι η πιο αποτελεσματική έχοντας και τους καλύτερους χρόνους στους υπολογισμούς που πραγματοποιήθηκαν. Αντίθετα, ο αλγόριθμος Bruteforce (Πολυπλοκότητα: O(n)) είναι ο πιο χρονοβόρος με μεγάλη διαφορά. Στην συνέχεια, ο βελτιωμένος αλγόριθμος Bruteforce (Πολυπλοκότητα: O(n/2)) φαίνεται να επιταχύνει την διαδικασία, αλλά ακόμα παραμένει αρκετά πιο αργός από τον αλγόριθμο της ρίζας. Τα αποτελέσματα αυτά είναι τα αναμενόμενα με βάση και την πολυπλοκότητα των αλγορίθμων αντίστοιχα.
Συγκρίνοντας όμως τώρα τις τρεις γλώσσες, Julia, Matlab και Python. Παρατηρούμε πολύ υψηλές ταχύτητες στην Julia , γεγονός που μας επιτρέπει δοκιμές με πολύ μεγάλες τιμές. Η διαφορά της με την Matlab είναι αισθητή αλλά δεν απαγορεύει τους υπολογισμούς με μεγαλύτερες τιμές για την τελευταία. Στην Python βλέπουμε μια σημαντική αύξηση στον χρόνο που χρειάστηκε για τους υπολογισμούς και παρά την ευκολία στην χρήση της δεν είναι επιθυμητό να προτιμάται για ανάλογους υπολογισμούς. Τέλος, όλες οι παραπάνω γλώσσες καταγράφουν συντριπτικά καλύτερους χρόνους από το Octave.