YILDIZ TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

Optimized Sieve of Eratosthenes in O(n) time complexity

Student Name Surname: Amirkia Rafiei Oskooei
Student Number: 19011919
E-mail: amirkia.oskooei@std.yildiz.edu.tr

Intro. to Structured Prog. BLM1012 Gr-2 Instructor: Lect. Ahmet ELBİR

June 2021

Introduction

In mathematics, Sieve of Eratosthenes is an old algorithm for finding all prime numbers up to any limit.

How it Works

In order to find all prime numbers less than or equal to a given integer n using the Eratosthenes **classic** method:

1. Create a list of consecutive integers from 2 to n
2. Initially, let p be equal to the smallest prime number 2.
3. 3. Count p multiples by counting p multiples from 2p to n and tick them off the list.
4. Find the smallest unsigned number in the list that is greater than p. Stop if there is no such number. Otherwise, set p to this new number (which is the next prime number) and repeat step 3 again.
5. When the algorithm completes, the unchecked numbers in the list are all prime numbers less than n.

Time Complexity: O(N log (log N))   *Memory Requirement: O(√n/log n)*

The **Manipulated** Eratosthenes Algorithm works as follows:

1. For each number i where i ranges from 2 to N-1:
   Check if the number is prime. If the number is prime, store it in the prime array.
2. For every prime numbers j less than or equal to the smallest prime factor p of i:
   a. Mark all numbers as j*p non-prime.
   b. Mark the smallest prime factor of j*p as j.

Time Complexity: O(N)   *Memory Requirement: O(√n log log n/log n)*

C Code:

```
C: > Users > HP > Desktop > C 19011919_AMIRKIARAFIEIOSKOOEI.c > manipulated_seive(int, int [], int [], int [], int *, int *)
//  program to generate all prime numbers less than N in O(N)
#include <stdio.h>
#define SIZE 200    // max size of array
#define upperLimit 170  // N

// protoypes
void manipulated_seive(int N, int isprime[], int prime[], int SPF[], int *primeIndex, int *counter);

// isPrime[] : isPrime[i] is true(1) if number is prime
// prime[] : stores all prime numbers less than N
// SPF[] that store smallest prime factor of number



int main()
{
    int i;
    int primeIndex = 0;     // current index of prime[] array
    int N = upperLimit;     // Must be less than MAX_SIZE
    int isprime[SIZE], prime[SIZE] , SPF[SIZE];
    int counter = 0;


    for ( i = 2; i < SIZE; i++)
    {
        isprime[i] = 1;     // initialize 1(True) to all elements
    }
    isprime[0] = 0;
    isprime[1] = 0;

    // envoke function
    manipulated_seive(N, isprime, prime, SPF, &primeIndex, &counter);

    // print all prime number less than N
    for ( i = 0; i < primeIndex; i++)
    {
        printf("%d /", prime[i]);
    }

    // print counter
    printf("\n\ncounter = %d\n", counter);



}
```

```c
// function generate all prime number less then N in O(n)
void manipulated_seive(int N, int isprime[], int prime[], int SPF[], int *primeIndex, int *counter)
{
    // 0 and 1 are not prime
    //isprime[0] = isprime[1] = false = 0 ;

    // Fill rest of the entries
    int i, j;

    for (i=2; i<N ; i++)
    {
        (*counter)++;

        // If isPrime[i] == True then i is prime number
        if (isprime[i] == 1)
        {
            // put i into prime[] array
            prime[*primeIndex] = i;
            (*primeIndex)++;
            // A prime number is its own smallest prime factor
            SPF[i] = i;
        }

        // Remove all multiples of  i*prime[j] which are
        // not prime by making isPrime[i*prime[j]] = false
        // and put smallest prime factor of i*Prime[j] as prime[j]
        // [ for exp :let  i = 5 , j = 0 , prime[j] = 2 [ i*prime[j] = 10 ]
        // so smallest prime factor of '10' is '2' that is prime[j] ]
        // this loop run only one time for number which are not prime
        for (j=0;j < (*primeIndex) && i*prime[j] < N && prime[j] <= SPF[i]; j++)
        {
            isprime[i*prime[j]]=0;

            // put smallest prime factor of i*prime[j]
            SPF[i*prime[j]] = prime[j] ;



        }
    }
}

```

Results:

```
#define upperLimit 125  // N
```

C:\WINDOWS\system32\cmd.exe

```
2 /3 /5 /7 /11 /13 /17 /19 /23 /29 /31 /37 /41 /43 /47 /53 /59 /61 /67 /71 /73 /79 /83 /89 /97 /101 /103 /107 /109 /113
/

counter = 123

Press any key to continue . . .
```

```
#define upperLimit 183  // N
```

C:\WINDOWS\system32\cmd.exe

```
2 /3 /5 /7 /11 /13 /17 /19 /23 /29 /31 /37 /41 /43 /47 /53 /59 /61 /67 /71 /73 /79 /83 /89 /97 /101 /103 /107 /109 /113
/127 /131 /137 /139 /149 /151 /157 /163 /167 /173 /179 /181 /

counter = 181

Press any key to continue . . .
```

Applications

- ✓ One of the most efficient approximations for prime numbers up to a few billions.
- ✓ Sieve of Eratosthenes is a popular way to compare computer performance.
- ✓ It is used for empirical studies of how prime numbers are distributed, which is a topic of great interest to analytical number theorists.

Competitors

We have three main Sieve algorithms for prime numbers:

- **SoE** (Sieve of Eratosthenes). The oldest method.
- **SoA** (Sieve of Atkin). The newest method. It has the best asymptotic complexity under certain assumptions.
- **The Sieve of Sundaram**. Interesting but not generally used.

Despite having a nice asymptotic complexity, **SoA** has a higher overhead in practice and requires some effort to implement properly. **SoE** is a better option when skill and time are invested equally.

Video Link (in Turkish)

https://youtu.be/ZSY33u2qNUs

References:

https://www.geeksforgeeks.org/sieve-of-eratosthenes/

https://www.geeksforgeeks.org/sieve-eratosthenes-0n-time-complexity/

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

https://wiki.haskell.org/Prime_numbers#Sieve_of_Eratosthenes

https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html