GeeksforGeeks

Related Articles Save for later

Segmented Sieve (Print Primes in a Range)

Difficulty Level : Medium • Last Updated : 06 Jun, 2021

Given a range [low, high], print all primes in this range? For example, if the given range is [10, 20], then output is 11, 13, 17, 19.

A **Naive approach** is to run a loop from low to high and check each number for primeness.
A Better Approach is to precalculate primes up to the maximum limit using Sieve of Eratosthenes, then print all prime numbers in range.
The above approach looks good, but consider the input range [50000, 55000]. the above Sieve approach would precalculate primes from 2 to 50100. This causes a waste of memory as well as time. Below is the Segmented Sieve based approach.

Segmented Sieve (Background)
Below are basic steps to get an idea of how Segmented Sieve works

1. Use Simple Sieve to find all primes up to a predefined limit (square root of 'high' is used in below code) and store these primes in an array "prime[]". Basically we call Simple Sieve for a limit and we not only find prime numbers, but also puts them in a separate array prime[].
2. Create an array mark[high-low+1]. Here we need only O(n) space where **n** is number of elements in given range.

3. Iterate through all primes found in step 1. For every prime, mark its multiples in given range [low..high].

So unlike simple sieve, we don't check for all multiples of every number smaller than square root of high, we only check for multiples of primes found in step 1. And we don't need O(high) space, we need O(sqrt(high) + n) space.
Below is the implementation of above idea.

C++

```
#include <bits/stdc++.h>
using namespace std;
// fillPrime function fills primes from 2 to sqrt of high in chprime(vect
void fillPrimes(vector<int>& prime, int high)
{
    bool ck[high + 1];
    memset(ck, true, sizeof(ck));
    ck[1] = false;
    ck[0] = false;
    for (int i = 2; (i * i) <= high; i++) {
        if (ck[i] == true) {
            for (int j = i * i; j <= high; j = j + i) {
                ck[j] = false;
            }
        }
    }
    for (int i = 2; i * i <= high; i++) {
        if (ck[i] == true) {
            prime.push_back(i);
        }
    }
}
// in segmented sieve we check for prime from range [low, high]
void segmentedSieve(int low, int high)
{
    bool prime[high - low + 1];
  //here prime[0] indicates whether low is prime or not similarly prime[1
    memset(prime, true, sizeof(prime));

    vector<int> chprime;
    fillPrimes(chprime, high);
    //chprimes has primes in range [2,sqrt(n)]
     // we take primes from 2 to sqrt[n] because the multiples of all pri
   // primes in range 2 to sqrt[n] for eg: for number 7 its multiples i.e
   // 28 is marked by 4, 35 is marked by 5, 42 is marked 6, so 49 will be
  // are marked by primes in range [2,sqrt(49)]
    for (int i : chprime) {
        int lower = (low / i);
        //here lower means the first multiple of prime which is in range
        //for eg: 3's first multiple in range [28,40] is 30
        if (lower <= 1) {
            lower = i + i;
        }
        else if (low % i) {
            lower = (lower * i) + i;
        }
        else {
            lower = (lower * i);
        }
        for (int j = lower; j <= high; j = j + i) {
            prime[j - low] = false;
        }
    }

    for (int i = low; i <= high; i++) {
        if (prime[i - low] == true) {
            cout << (i) << " ";
        }
    }
}
int main()
{
    // low should be greater than or equal to 2
    int low = 2;
    // low here is the lower limit
    int high = 100;
    // high here is the upper limit
      // in segmented sieve we calculate primes in range [low,high]
   // here we initially we find primes in range [2,sqrt(high)] to mark al
   //then we mark all their(primes) multiples in range [low,high] as fals
   // this is a modified sieve of eratosthenes , in standard sieve of  er
   // in segmented sieve we only find primes in a given interval
  cout<<"Primes in range "<<low<<" to "<< high<<" are\n";
    segmentedSieve(low, high);
}
```

Python

```
import math

# fillPrime function fills primes from 2 to sqrt of high in chprime list
def fillPrimes(chprime, high):

    ck = [True]*(high+1)

    l = int(math.sqrt(high))
    for i in range(2, l+1):
        if ck[i]:
            for j in range(i*i, l+1, i):
                ck[j] = False



    for k in range(2, l+1):
        if ck[k]:
            chprime.append(k)

# in segmented sieve we check for prime from range [low, high]
def segmentedSieve(low, high):

    chprime = list()
    fillPrimes(chprime, high)
# chprimes has primes in range [2,sqrt(n)]
# we take primes from 2 to sqrt[n] because the multiples of all primes be
# primes in range 2 to sqrt[n] for eg: for number 7 its multiples i.e 14
# 28 is marked by 4, 35 is marked by 5, 42 is marked 6, so 49 will be fir
# are marked by primes in range [2,sqrt(49)]
    prime = [True] * (high-low + 1)
# here prime[0] indicates whether low is prime or not similarly prime[1]
    for i in chprime:
        lower = (low//i)
# here lower means the first multiple of prime which is in range [low,hig
# for eg: 3's first multiple in range [28,40] is 30
        if lower <= 1:
            lower = i+i
        elif (low % i) != 0:
            lower = (lower * i) + i
        else:
            lower = lower*i
            for j in range(lower, high+1, i):
                prime[j-low] = False


        for k in range(low, high + 1):
                if prime[k-low]:
                    print(k, end=" ")


#DRIVER CODE
#   low should be greater than or equal to 2
low = 2
#  low here is the lower limit
high = 100
# high here is the upper limit
# in segmented sieve we calculate primes in range [low,high]
# here we initially we find primes in range [2,sqrt(high)] to mark all th
# then we mark all their(primes) multiples in range [low,high] as false
# this is a modified sieve of eratosthenes , in standard sieve of  eratos
# in segmented sieve we only find primes in a given interval
print("Primes in Range %d to %d are"%(low,high))
segmentedSeive(low, high)
```

Java

```
import java.util.*;

public class Main{
// fillPrime function fills primes from 2 to sqrt of high in chprime Arra
    public static void fillPrime(ArrayList<Integer> chprime,int high)
    {
        boolean[] ck=new boolean[high+1];
        Arrays.fill(ck,true);
        ck[1]=false;
        ck[0]=false;

    for(int i=2;(i*i)<=high;i++)
    {
        if(ck[i]==true)
        {
            for(int j=i*i;j<=high;j=j+i)
            {
                ck[j]=false;
            }
        }
    }
    for(int i=2;i*i<=high;i++)
    {
        if(ck[i]==true)
        {
   //          cout<< i<<"\n";
           chprime.add(i);
        }
    }
}
// in segmented sieve we check for prime from range [low, high]
    public static void segmentedSieve(int low,int high)
    {
        ArrayList<Integer> chprime= new ArrayList<Integer>();
        fillPrime(chprime,high);
//chprimes has primes in range [2,sqrt(n)]
// we take primes from 2 to sqrt[n] because the multiples of all primes b
// primes in range 2 to sqrt[n] for eg: for number 7 its multiples i.e 14
// 28 is marked by 4, 35 is marked by 5, 42 is marked 6, so 49 will be fi
// are marked by primes in range [2,sqrt(49)]

        boolean[] prime=new boolean [high-low+1];
        Arrays.fill(prime,true);
//here prime[0] indicates whether low is prime or not similarly prime[1]
        for(int i:chprime)
        {
            int lower=(low/i);
//here lower means the first multiple of prime which is in range [low,hig
//for eg: 3's first multiple in range [28,40] is 30
            if(lower<=1)
            {
            lower=i+i;
            }
            else if(low%i!=0)
            {
            lower=(lower*i)+i;
            }
            else{
                lower=(lower*i);
            }
            for(int j=lower;j<=high;j=j+i)
            {
            prime[j-low]=false;
            }
        }
        for(int i=low;i<=high;i++)
        {
            if(prime[i-low]==true)
            {
            System.out.printf("%d ",i);
            }
        }
    }
    public static void main(String[] args)
    {
// low should be greater than or equal to 2
        int low=2;
// low here is the lower limit
        int high=100;
// high here is the upper limit
// in segmented sieve we calculate primes in range [low,high]
// here we initially we find primes in range [2,sqrt(high)] to mark all t
//then we mark all their(primes) multiples in range [low,high] as false
// this is a modified sieve of eratosthenes , in standard sieve of  erato
// in segmented sieve we only find primes in a given interval
        System.out.println("Primes in Range "+low+" to "+high+" are ");
        segmentedSieve(low,high);
    }
}
```

Output

```
11  13  15  17  19
```

Segmented Sieve (What if 'high' value of range is too high and range is also big)

Consider a situation where given high value is so high that neither sqrt(high) nor O(high-low+1) can fit in memory. How to find prims in range. For this situation, we run step 1 (Simple Sieve) only for a limit that can fit in memory. Then we divide given range in different segments. For every segment, we run step 2 and 3 considering low and high as end points of current segment. We add primes of current segment to prime[] before running the next segment.

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Like 22

Previous

Segmented Sieve

Next

Prime Factorization using Sieve O(log n) for multiple queries

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Segmented Sieve**
24, Nov 15

02 **Longest sub-array of Prime Numbers using Segmented Sieve**
26, Mar 20

03 **Sum of all Primes in a given range using Sieve of Eratosthenes**

05 **Nth Term of a Fibonacci Series of Primes formed by concatenating pairs of Primes in a given range**
07, Aug 20

06 **Count primes that can be expressed as sum of two consecutive primes and 1**
12, Feb 19

07 **Count of primes below N which can be expressed as the sum of**

17, Oct 18

two primes
09, Dec 19

04 **Sieve of Sundaram to print all primes smaller than n**
12, Jan 16

08 **K-Primes (Numbers with k prime factors) in a range**
05, Dec 17

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : RajatSinghal, nikhilaggarwal3, gauravak007, deveshanand18, aditya04848, jyoti369, skipper13, aditya1762002, saurabh1990aror

Article Tags : Mathematical

Practice Tags : Mathematical

Improve Article Report Issue