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

We can calculate the prime factorization of a number **"n"** in **O(sqrt(n))** as discussed here. But O(sqrt n) method times out when we need to answer multiple queries regarding prime factorization.

In this article we study an efficient method to calculate the prime factorization using O(n) space and **O(log n)** time complexity with per-computation allowed.

Prerequisites : Sieve of Eratosthenes, Least prime factor of numbers till n.

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Key Concept: *Our idea is to store the Smallest Prime Factor(SPF) for every number. Then to calculate the prime factorization of the given number by dividing the given number recursively with its smallest prime factor till it becomes 1.*

To calculate to smallest prime factor for every number we will use the sieve of eratosthenes. In original Sieve, every time we mark a number as not-prime, we store the corresponding smallest prime factor for that number (Refer this article for better understanding).

Now, after we are done with precalculating the smallest prime factor for every number we will divide our number n (whose prime factorziation is to be calculated) by its corresponding smallest pri factor till n becomes 1.

```
PrimeFactors[] // To store result

i = 0  // Index in PrimeFactors

while n != 1 :

    // SPF : smallest prime factor
    PrimeFactors[i] = SPF[n]
    i++
    n = n / SPF[n]
```

The implementation for the above method is given below :

C++

```cpp
// C++ program to find prime factorization of a
// number n in O(Log n) time with precomputation
// allowed.
#include "bits/stdc++.h"
using namespace std;

#define MAXN   100001

// stores smallest prime factor for every number
int spf[MAXN];

// Calculating SPF (Smallest Prime Factor) for every
// number till MAXN.
// Time Complexity : O(nloglogn)
void sieve()
{
    spf[1] = 1;
    for (int i=2; i<MAXN; i++)

        // marking smallest prime factor for every
        // number to be itself.
        spf[i] = i;

    // separately marking spf for every even
    // number as 2
    for (int i=4; i<MAXN; i+=2)
        spf[i] = 2;

    for (int i=3; i*i<MAXN; i++)
    {
        // checking if i is prime
        if (spf[i] == i)
```

```
                // marking spf[j] if it is not
                // previously marked
                if (spf[j]==j)
                    spf[j] = i;
        }
    }
}

// A O(log n) function returning primefactorization
// by dividing by smallest prime factor at every step
vector<int> getFactorization(int x)
{
    vector<int> ret;
    while (x != 1)
    {
        ret.push_back(spf[x]);
        x = x / spf[x];
    }
    return ret;
}

// driver program for above function
int main(int argc, char const *argv[])
{
    // precalculating Smallest Prime Factor
    sieve();
    int x = 12246;
    cout << "prime factorization for " << x << " : ";

    // calling getFactorization function
    vector <int> p = getFactorization(x);

    for (int i=0; i<p.size(); i++)
        cout << p[i] << " ";
    cout << endl;
    return 0;
}
```

Java

```
// Java program to find prime factorization of a
// number n in O(Log n) time with precomputation
// allowed.

import java.util.Vector;

class Test
{
    static final int MAXN = 100001;
    // number till MAXN.
    // Time Complexity : O(nloglogn)
    static void sieve()
    {
        spf[1] = 1;
        for (int i=2; i<MAXN; i++)

            // marking smallest prime factor for every
            // number to be itself.
            spf[i] = i;

        // separately marking spf for every even
        // number as 2
        for (int i=4; i<MAXN; i+=2)
            spf[i] = 2;

        for (int i=3; i*i<MAXN; i++)
        {
            // checking if i is prime
            if (spf[i] == i)
            {
                // marking SPF for all numbers divisible by i
                for (int j=i*i; j<MAXN; j+=i)

                    // marking spf[j] if it is not
                    // previously marked
                    if (spf[j]==j)
                        spf[j] = i;
            }
        }
    }

    // A O(log n) function returning primefactorization
    // by dividing by smallest prime factor at every step
    static Vector<Integer> getFactorization(int x)
    {
        Vector<Integer> ret = new Vector<>();
        while (x != 1)
        {
            ret.add(spf[x]);
            x = x / spf[x];
        }
        return ret;
    }

    // Driver method
    public static void main(String args[])
    {
        // precalculating Smallest Prime Factor
        sieve();
        int x = 12246;
        System.out.print("prime factorization for " + x + " : ");
        for (int i=0; i<p.size(); i++)
            System.out.print(p.get(i) + " ");
        System.out.println();
    }
}
```

Python3

```
# Python3 program to find prime factorization
# of a number n in O(Log n) time with
# precomputation allowed.
import math as mt

MAXN = 100001

# stores smallest prime factor for
# every number
spf = [0 for i in range(MAXN)]

# Calculating SPF (Smallest Prime Factor)
# for every number till MAXN.
# Time Complexity : O(nloglogn)
def sieve():
    spf[1] = 1
    for i in range(2, MAXN):

        # marking smallest prime factor
        # for every number to be itself.
        spf[i] = i

    # separately marking spf for
    # every even number as 2
    for i in range(4, MAXN, 2):
        spf[i] = 2

    for i in range(3, mt.ceil(mt.sqrt(MAXN))):

        # checking if i is prime
        if (spf[i] == i):

            # marking SPF for all numbers
            # divisible by i
            for j in range(i * i, MAXN, i):

                # marking spf[j] if it is
                # not previously marked
                if (spf[j] == j):
                    spf[j] = i

# A O(log n) function returning prime
```

```
    while (x != 1):
        ret.append(spf[x])
        x = x // spf[x]

    return ret

# Driver code

# precalculating Smallest Prime Factor
sieve()
x = 12246
print("prime factorization for", x, ": ",
                                end = "")

# calling getFactorization function
p = getFactorization(x)

for i in range(len(p)):
    print(p[i], end = " ")

# This code is contributed
# by Mohit kumar 29
```

C#

```
// C# program to find prime factorization of a
// number n in O(Log n) time with precomputation
// allowed.
using System;
using System.Collections;

class GFG
{
    static int MAXN = 100001;

    // stores smallest prime factor for every number
    static int[] spf = new int[MAXN];

    // Calculating SPF (Smallest Prime Factor) for every
    // number till MAXN.
    // Time Complexity : O(nloglogn)
    static void sieve()
    {
        spf[1] = 1;
        for (int i = 2; i < MAXN; i++)

            // marking smallest prime factor for every
            // number to be itself.
            spf[i] = i;
        for (int i = 3; i * i < MAXN; i++)
        {
            // checking if i is prime
            if (spf[i] == i)
            {
                // marking SPF for all numbers divisible by i
                for (int j = i * i; j < MAXN; j += i)

                    // marking spf[j] if it is not
                    // previously marked
                    if (spf[j] == j)
                        spf[j] = i;
            }
        }
    }

    // A O(log n) function returning primefactorization
    // by dividing by smallest prime factor at every step
    static ArrayList getFactorization(int x)
    {
        ArrayList ret = new ArrayList();
        while (x != 1)
        {
            ret.Add(spf[x]);
            x = x / spf[x];
        }
        return ret;
    }

    // Driver code
    public static void Main()
    {
        // precalculating Smallest Prime Factor
        sieve();
        int x = 12246;
        Console.Write("prime factorization for " + x + " : ");

        // calling getFactorization function
        ArrayList p = getFactorization(x);

        for (int i = 0; i < p.Count; i++)
            Console.Write(p[i] + " ");
        Console.WriteLine("");
    }
}

// This code is contributed by mits
```

PHP

```
// precomputation allowed.

$MAXN = 19999;

// stores smallest prime factor for
// every number
$spf = array_fill(0, $MAXN, 0);

// Calculating SPF (Smallest Prime Factor)
// for every number till MAXN.
// Time Complexity : O(nloglogn)
function sieve()
{
    global $MAXN, $spf;
    $spf[1] = 1;
    for ($i = 2; $i < $MAXN; $i++)

        // marking smallest prime factor
        // for every number to be itself.
        $spf[$i] = $i;

    // separately marking spf for every
    // even number as 2
    for ($i = 4; $i < $MAXN; $i += 2)
        $spf[$i] = 2;

    for ($i = 3; $i * $i < $MAXN; $i++)
    {
        // checking if i is prime
        if ($spf[$i] == $i)
        {
            // marking SPF for all numbers
            // divisible by i
            for ($j = $i * $i; $j < $MAXN; $j += $i)

                // marking spf[j] if it is not
                // previously marked
                if ($spf[$j] == $j)
                    $spf[$j] = $i;
        }
    }
}

// A O(log n) function returning primefactorization
// by dividing by smallest prime factor at every step
function getFactorization($x)
{
    global $spf;
    $ret = array();
    while ($x != 1)
    {
        array_push($ret, $spf[$x]);
        if($spf[$x])
}

// Driver Code

// precalculating Smallest
// Prime Factor
sieve();
$x = 12246;
echo "prime factorization for " .
                      $x . " : ";

// calling getFactorization function
$p = getFactorization($x);

for ($i = 0; $i < count($p); $i++)
    echo $p[$i] . " ";

// This code is contributed by mits
?>
```

Output:

```
prime factorization for 12246 : 2 3 13 157
```

Note : The above code works well for n upto the order of 10^7. Beyond this we will face memory issues.

Time Complexity: The precomputation for smallest prime factor is done in O(n log log n) using sieve. Where as in the calculation step we are dividing the number every time by the smallest prime number till it becomes 1. So, let's consider a worst case in which every time the SPF is 2 . Therefore will have log n division steps. Hence, We can say that our Time Complexity will be **O(log n)** in worst case.

This article is contributed by **Nitish Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Sum of Factors of a Number using Prime Factorization

Pollard's Rho Algorithm for Prime Factorization

Queries on the sum of prime factor counts in a range

Find First element in AP which is multiple of given prime

Queries for maximum difference between prime numbers in given ranges

Queries to count the number of unordered co-prime pairs from 1 to N

Sieve of Atkin

Segmented Sieve

Bitwise Sieve

Sieve of Eratosthenes

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

Sieve of Eratosthenes in 0(n) time complexity

Segmented Sieve (Print Primes in a Range)

Number of unmarked integers in a special sieve

Sieve of Sundaram to print all primes smaller than n

Improved By : mohit kumar 29, Mithun Kumar

Article Tags : Mathematical prime-factor sieve

Practice Tags : Mathematical sieve

3

To-do Done

3

Based on **28** vot

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!