← Notes

Number Theory - III

63 CodeMonk Euler's totient function Sieve of Eratosthenes

Euler's totient function

Description: **phi(N)** counts the number of integers from **1** to **N** inclusive that are relatively prime to **N**.

Implemention: let me remind you that *factorization* is the way to represent given number as a product of primes. And it's easy to see that for every number such representation is unique. For example:

8 = 2^3

11 = 11

36 = 2^2 * 3^2

935 = 5 * 11 * 17

5136 = 2^4 * 3 * 107

So, implemention is based on factorization:

```
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res -= res / i;
        }
    }
    if (n != 1) {
        res -= res / n;
    }
    return res;
}
```

It works in **O(sqrt(N))** time. How to make it faster read further.

Modification of Sieve of Eratosthenes for fast factorization

Implemention: let's see how we can factorize **N** in **O(sqrt(N))** time

```
vector<int> factorize(int n) {
    vector<int> res;
    for (int i = 2; i * i <= n; ++i) {
        while (n % i == 0) {
            res.push_back(i);
            n /= i;
        }
    }
    if (n != 1) {
        res.push_back(n);
    }
    return res;
}
```

At every step we are looking for a minimal prime number that divides our current **N**. This is the main idea of Sieve's modification. Let's construct such array which in **O(1)** time will give us this number:

```
int minPrime[n + 1];
for (int i = 2; i * i <= n; ++i) {
    if (minPrime[i] == 0) { //if i is prime
        for (int j = i * i; j <= n; j += i) {
            if (minPrime[j] == 0) {
                minPrime[j] = i;
            }
        }
    }
}
for (int i = 2; i <= n; ++i) {
    if (minPrime[i] == 0) {
        minPrime[i] = i;
    }
}
```

Now we can factorize **N** in **O(log(N))** time using this modification:

```
vector<int> factorize(int n) {
    vector<int> res;
    while (n != 1) {
        res.push_back(minPrime[n]);
        n /= minPrime[n];
    }
```

```
    return res;
}
```

Conditions: you can implement this modification only if you're allowed to create an array of integers with size **N**.

Advices: this approach is useful when you need to factorize a lot of times some not very large numbers. It's not necessary to build such modified Sieve in every problem where you need to factorize something. Moreover you can't build it for such large **N** like 10^9 or 10^{12}. So, use factorization in **O(sqrt(N))** instead.

Cool fact: if factorization of **N** is $p_1^{q_1} * p_2^{q_2} * ... * p_k^{q_k}$ then **N** has $(q_1 + 1) * (q_2 + 1) * ... * (q_k + 1)$ distinct divisors.

Sieve of Eratosthenes on the segment

Sometimes you need to find all primes not in range **[1...N]** but in range **[L...R]**, where **R** is large enough.

Conditions: you're allowed to create an array of integers with size **(R - L + 1)**.

Implemention:

```
bool isPrime[r - l + 1]; //filled by true
for (long long i = 2; i * i <= r; ++i) {
    for (long long j = max(i * i, (l + (i - 1)) / i * i); j <= r; j
+= i) {
        isPrime[j - l] = false;
    }
}
for (long long i = max(l, 2); i <= r; ++i) {
    if (isPrime[i - l]) {
        //then i is prime
    }
}
```

The approximate comlexity is **O(sqrt(R) * const)**

Advices: again it's not necessary to build such Sieve if you need to check just several numbers for primality. Use the following function instead which works in **O(sqrt(N))** for every number:

```
bool isPrime(int n) {
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            return false;
```

```
            }
        }
        return true;
    }
```


COMMENTS (19) SORT BY: Relevance

Join Discussion...

Cancel Post

Vikrant Mahriya 2 years ago

"At every step we are looking for a minimal prime number that divides our current N."
We are getting minPrime[6]=2 which is correct by definition.
But why minPrime[12]=3 ? Shouldn't it be 2 ?

0 votes ● Reply ● Message ● Permalink

Boris Sokolov Author 2 years ago

You're right, sometimes minPrime[i] stores not the minimal prime divisor of i, but for complexity of factorization it doesn't matter :) If you wanna store exactly minimal prime divisor you can add "if (minPrime[j] == 0)" before "minPrime[j] = i".

0 votes ● Reply ● Message ● Permalink

Vikrant Mahriya 2 years ago

Yes. Thanks :)

1 vote ● Reply ● Message ● Permalink

hellman h 2 years ago

then the variable should called differenrtly, e.g. someFactor?

1 vote ● Reply ● Message ● Permalink

Sushma Kumari 2 years ago

You can include this Youtube video link : https://www.youtube.com/watch?v=GZbdmCIhmpA
Coding is easy once the idea is clear in anyone's head :)

1 vote ● Reply ● Message ● Permalink

Vignesh Mahalingam 2 years ago

Can you please add in-line comments to the code for better understanding?
And thank you for this notes.

0 votes ● Reply ● Message ● Permalink

ha 2 years ago

Why is it so that if we select any n regularly spaced integers then
there would be exactly one integer among them which would be
divisible by n ?
note: the common difference should not be n.

0 votes ● Reply ● Message ● Permalink

Boris Sokolov Author 2 years ago