


[HOME](#) [TOP](#) [CONTESTS](#) [GYM](#) [PROBLEMSET](#) [GROUPS](#) [RATING](#) [API](#) [HELP](#) [KOTLIN HEROES](#)  [DASHA](#)  [CALENDAR](#)

 ⓘ Your password is extremely weak or has been leaked (see <https://haveibeenpwned.com/>). Please, [change it](#) ASAP.

[NISIYAMA_SUZUNE](#) [BLOG](#) [TEAMS](#) [SUBMISSIONS](#) [GROUPS](#) [CONTESTS](#)

Nisiyama_Suzune's blog

[Tutorial] Math note — linear sieve

 By [Nisiyama_Suzune](#), 2 years ago, 

This short article is mainly focused on the linear sieve and its various applications in competitive programming, including a brief introduction on how to pick out primes and a way to calculate multiple values of multiplicative functions.

Sieve of Eratosthenes

While this name may sound scary, the sieve of Eratosthenes is probably the simplest way to pick out all the primes in a given range from 1 to n . As we already know, one of the properties that all primes have is that they do not have any factors except 1 and themselves. Therefore, if we cross out all composites, which have at least one factor, we can obtain all primes. The following code demonstrates a simple implementation of the said algorithm:

```
std::vector<int> prime;
bool is_composite[MAXN];

void sieve (int n) {
    std::fill (is_composite, is_composite + n, false);
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) prime.push_back (i);
        for (int j = 2; i * j < n; ++j)
            is_composite[i * j] = true;
    }
}
```

As we can see, the statement `is_composite[i * j] = true;` crosses out all numbers that do have a factor, as they are all composites. All remaining numbers, therefore, should be prime. We then check for those primes and put them into a container named `prime`.


Linear sieve

It can be analyzed that the method above runs in $O(n \log n)$ complexity (with the Euler–Mascheroni constant, i.e. $\lim_{n \rightarrow \infty} \sum_{i=1}^n \frac{1}{i} = \ln n + \gamma$). Let us take a minute to consider the bottleneck of such sieve. While we do need to cross out each composite once, in practice we run the inner loop for a composite multiple times due to the fact that it has multiple factors. Thus, if we can establish a unique representation for each composite and pick them out only once, our algorithm will be somewhat better. Actually it is possible to do so. Note that every composite q must have at least one prime factor, so we can pick the smallest prime factor p , and let the rest of the part be i , i.e. $q = ip$. Since p is the smallest prime factor, we have $i \geq p$, and no prime less than p can divide i . Now let us take a look at the code we have a moment ago. When we loop for every i , all primes not exceeding i is already recorded in the container `prime`. Therefore, if we only loop for all elements in `prime` in the inner loop, breaking out when the element divides i , we can pick out each composite exactly once.

→ Pay attention

Before contest

[Codeforces Round #584 - Dasha Code Championship - Elimination Round \(rated, open for everyone, Div. 1 + Div. 2\)](#)
00:59:27

 299 people like this. [Sign Up](#) to see what your friends like.

→ s_h_shahin

Rating: **1858**
 Contribution: 0
[Settings](#)
[Blog](#)
[Teams](#)
[Submissions](#)
[Problemsetting](#)
[Groups](#)
[Propose a contest/problems](#)
[Talks](#)
[Contests](#)

 [s_h_shahin](#)

→ Top rated

#	User	Rating
1	tourist	3711
2	wxhtxdy	3463
3	Um_nik	3419
4	Radewoosh	3344
5	LHIC	3336
6	Benq	3316
7	yutaka1999	3190
8	ainta	3180
9	mnbvmar	3128
10	Petr	3106

[Countries](#) | [Cities](#) | [Organizations](#)
[View all →](#)

→ Top contributors

#	User	Contrib.
1	Errichto	192
2	Radewoosh	173
3	PikMike	165
4	Vovuh	161
4	rng_58	161

```

std::vector<int> prime;
bool is_composite[MAXN];

void sieve (int n) {
    std::fill (is_composite, is_composite + n, false);
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) prime.push_back (i);
        for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) break;
        }
    }
}

```

As is shown in the code, the statement `if (i % prime[j] == 0) break;` terminates the loop when p divides i . From the analysis above, we can see that the inner loop is executed only once for each composite. Hence, the code above performs in $O(n)$ complexity, resulting in its name — the 'linear' sieve.

Multiplicative function

There is one specific kind of function that shows importance in the study of number theory — the multiplicative function. By definition, A function $f(x)$ defined on all positive integers is multiplicative if it satisfies the following condition:

For every co-prime pair of integers p and q , $f(pq) = f(p)f(q)$.

Applying the definition, it can be shown that $f(n) = f(n)f(1)$. Thus, unless for every integer n we have $f(n) = 0$, $f(1)$ must be 1. Moreover, two multiplicative functions $f(n)$ and $g(n)$ are identical if and only if for every prime p and non-negative integer k , $f(p^k) = g(p^k)$ holds true. It can then be implied that for a multiplicative function $f(n)$, it will suffice to know about its representation in $f(p^k)$.

The following functions are more or less commonly used multiplicative functions, according to Wikipedia:

1. The constant function $I(p^k) = 1$.
2. The power function $Id_a(p^k) = p^{ak}$, where a is constant.
3. The unit function $\varepsilon(p^k) = [p^k = 1]$ ($[P]$ is 1 when P is true, and 0 otherwise).
4. The divisor function $\sigma_a(p^k) = \sum_{i=0}^k p^{ai}$, denoting the sum of the a -th powers of all the positive divisors of the number.
5. The Möbius function $\mu(p^k) = [k = 0] - [k = 1]$.
6. The Euler's totient function $\varphi(p^k) = p^k - p^{k-1}$.

It is interesting that the linear sieve can also be used to find out all the values of a multiplicative function $f(x)$ in a given range $[1, n]$. To do so, we have take a closer look in the code of the linear sieve. As we can see, every integer x will be picked out only once, and we must know one of the following property:

1. x is prime. In this case, we can determine the value of $f(x)$ directly.
2. $x = ip$ and p does not divide i . In this case, we know that $f(x) = f(i)f(p)$. Since both $f(i)$ and $f(p)$ are already known before, we can simply multiply them together.
3. $x = ip$ and p divides i . This is a more complicated case where we have to figure out a relationship between $f(i)$ and $f(ip)$. Fortunately, in most situations, a simple relationship between them exists. For example, in the Euler's totient function, we can easily infer that $\varphi(ip) = p\varphi(i)$.

Since we can compute the function value of x satisfying any of the above property, we can simply modify the linear sieve to find out all values of the multiplicative function from 1 to n . The following code implements an example on the Euler's totient function.

6	majk	157
6	300iq	157
8	antontrygubO_o	156
9	Um_nik	150
10	kostka	148

[View all →](#)

→ Find user

Handle:

→ Recent actions

Errichto → [Codeforces Round #584 \(Dasha Code Championship Elimination Round\) \(div. 1 + div. 2\)](#)

dunjen_master → [Help in a question](#)

Arpa → [Call for problem setting on HackerEarth](#)

DSiyuan → [Auto SEO](#)

DSiyuan → [Welcome](#)

DSiyuan → [%Siyuan](#)

Vovuh → [Codeforces Round #582 \(Div. 3\) Editorial](#)

djm03178 → [Codeforces Round #578 \(Div. 2\) Editorial](#)

Arpa → [Data Structures and Algorithms coding challenge #2](#)

usaxena95 → [SOS Dynamic Programming \[Tutorial\]](#)

typedeftemplate → [Linear time algorithm to rank elements with memory constraints](#)

Noob-ita-pro → [Query Problem](#)

Theo830 → [Can someone help?](#)

BledDest → [Codeforces Round #585 \(Div. 2, based on Qualification Stage of Southern and Volga Russia Regional Contest\)](#)

Biza → [Mirror of Bubble Cup 12 Finals on Codeforces](#)

tourist → [Hello 2018](#)

300iq → [Codeforces Round #562 — Editorial](#)

PikMike → [Educational Codeforces Round 72 Editorial](#)

300iq → [XX Open Cup: GP of Kazan](#)

roosephu → [ZeptoLab Code Rush 2015 — Editorial](#)

kuangben → [About programming websites](#)

hmehta → [Topcoder SRM 766](#)

one0ne → [Python Code Difference of Two Solutions](#)

removed1 → [Codeforces Beta Round #1 - Tutorial](#)

powerHouseOfTheCell → [Segment Tree of DSUs](#)

[Detailed →](#)

```

std::vector<int> prime;
bool is_composite[MAXN];
int phi[MAXN];

void sieve (int n) {
    std::fill (is_composite, is_composite + n, false);
    phi[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back (i);
            phi[i] = i - 1;           // i is
prime
        }
        for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                //prime[j] divides i
                break;
            } else {
                phi[i * prime[j]] = phi[i] * phi[prime[j]];
                //prime[j] does not divide i
            }
        }
    }
}

```

Extension on the linear sieve

Well, it might not always be possible to find out a formula when p divides i . For instance, I can write some random multiplicative function $f(p^k) = k$ which is difficult to infer a formula with. However, there is still a way to apply the linear sieve on such function. We can maintain a counter array $cnt[i]$ denoting the power of the smallest prime factor of i , and find a formula using the array since $f(ip) = f(\frac{i}{p^{cnt[i]}})f(p^{cnt[i]+1})$. The following code gives an example on the function I've just written.

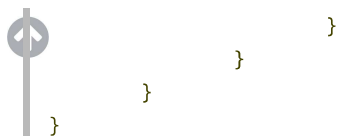
```

std::vector<int> prime;
bool is_composite[MAXN];
int func[MAXN], cnt[MAXN];

void sieve (int n) {
    std::fill (is_composite, is_composite + n, false);
    func[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back (i);
            func[i] = 1; cnt[i] = 1;
        }
        for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                func[i * prime[j]] = func[i] / cnt[i] * (cnt[i]
+ 1);

                cnt[i * prime[j]] = cnt[i] + 1;
                break;
            } else {
                func[i * prime[j]] = func[i] * func[prime[j]];
                cnt[i * prime[j]] = 1;
            }
        }
    }
}

```



Example problems

The Embarrassed Cryptographer

[Link](#)

Solution: Simply applying the linear sieve will be enough.

Farey Sequence

[Link](#)

Solution: Notice that $\Delta F(n) = F(n) - F(n-1)$ is multiplicative and $\Delta F(p^k) = p^k - p^{k-1}$. Then the linear sieve will come in handy to solve the problem.

math, sieve, tutorial

+226



[Nisiyama_Suzune](#)

2 years ago

44



Comments (44)

[Write comment?](#)



I_LOVE_METSUKA

2 years ago, # | ☆

+2

I don't understand anything. But it looks like something useful so upvote for you.

→ [Reply](#)



gilcu3

2 years ago, # | ☆

0

@Nisiyama_Suzune do you have any origin reference for this linear sieve? It's amazing it's not well known (as far I know) in the competitive programming world.

→ [Reply](#)



Nisiyama_Suzune

2 years ago, # ^ | ☆

+23

It would appear that this technique is established in 1978 by David Gries and Jayadev Misra at the communication of the ACM. They had a paper [here](#). However, I was unable to purchase it so I cannot determine whether it is true.

→ [Reply](#)



gilcu3

2 years ago, # ^ | ☆

0

Yes I have just checked :) Good source...

→ [Reply](#)



Fischer

2 years ago, # ^ | ☆

0

A linear sieve algorithm for finding prime numbers *- great post.

→ [Reply](#)

2 years ago, # ^ | ☆

+5

Thanks for the compliment~\