Sieve of Eratosthenes

Sieve of Eratosthenes - an algorithm for finding all primes in the interval $[1; n]$ for $O(n \log \log n)$ operations.

The idea is simple - to write a series of numbers $1 \ldots n$, and will be first to cross out all numbers divisible by 2, except for the numbers 2, then dividing by 3, except for the number 3, then on 5, then 7, 11 and all other simple to n.

Implementation

Immediately give the implementation of the algorithm:

```
int n;
vector<char> prime (n+1, true);
prime[0] = prime[1] = false;
for (int i=2; i<=n; ++i)
     if (prime[i])
          if (i * 1ll * i <= n)
               for (int j=i*i; j<=n; j+=i)
                    prime[j] = false;
```

This code first checks all numbers except zero and one, as simple, and then begins the process of sifting composite numbers. To do this, we loop through all the numbers 2 up n, and if the current number i is prime, mark all multiples of him as constituents.

At the same time we start walking from i^2 as fewer multiples i necessarily have a prime divisor less i, which means that they have already been eliminated before.(But as i^2 can easily overwhelm type int in the code before the second nested loop is an additional check using the type $long\ long$.)

With this implementation, the algorithm consumes $O(n)$ memory (obviously) and performs the $O(n \log \log n)$ action (this is proved in the next section).

Asymptotics

We prove that the asymptotic behavior of the algorithm is $O(n \log \log n)$.

So, for each prime $p \leq n$ inner loop will be executed, which make $\frac{n}{p}$ actions. Consequently, we need to estimate the following value:

$$\sum_{\substack{p\le n,\\ p\ is\ prime}} \frac{n}{p} = n \cdot \sum_{\substack{p\le n,\\ p\ is\ prime}} \frac{1}{p}.$$

Let us recall here two known facts: that the number of primes less than or equal to n approximately equal $\frac{n}{\ln n}$, and that kth prime number is approximately equal to $k \ln k$ (this follows from the first statement). Then the sum can be written as follows:

$$\sum_{\substack{p\le n,\\ p\ is\ prime}} \frac{1}{p} \approx \frac{1}{2} + \sum_{k=2}^{\frac{n}{\ln n}} \frac{1}{k \ln k}.$$

Here we have identified the first prime of the sum, since in $k = 1$accordance with the approximation $k \ln k$will $0$that lead to division by zero.

We now estimate that amount by the integral of the same function on $k$from $2$before $\frac{n}{\ln n}$ (we can produce such an approximation, because, in fact, refers to the amount of his integral approximation formula of rectangles):

$$\sum_{k=2}^{\frac{n}{\ln n}} \frac{1}{k \ln k} \approx \int_2^{\frac{n}{\ln n}} \frac{1}{k \ln k}\ dk.$$

Primitive of the integrand there $\ln \ln k$. Performing substitution and removing members of the lower order, we get:

$$\int_2^{\frac{n}{\ln n}} \frac{1}{k \ln k}\ dk = \ln \ln \frac{n}{\ln n} - \ln \ln 2 = \ln(\ln n - \ln \ln n) - \ln \ln 2 \approx \ln \ln n.$$

Now, returning to the initial sum, we obtain an approximate estimate of its:

$$\sum_{\substack{p\le n,\\ p\ is\ prime}} \frac{n}{p} \approx n \ln \ln n + o(n),$$

QED.

More rigorous proof (and provide a more accurate estimate of up to constant factors) can be found in the book of Hardy and Wright "An Introduction to the Theory of Numbers" (p. 349).

Various optimizations sieve of Eratosthenes

The biggest drawback of the algorithm - that he "walks" from memory, constantly going beyond the cache, causing the constant hidden in the $O(n \log \log n)$relatively large.

Moreover, for sufficiently large na bottleneck memory usage.

The following are the methods to both reduce the number of operations performed, and significantly reduce memory consumption.

Sifting simple to the root

The most obvious point - that in order to find all simple to $n$sufficiently perform only simple sieving not exceeding the root of n.

Thus, change the outer loop of the algorithm:

```
for (int i=2; i*i<=n; ++i)
```

On the asymptotic behavior of this optimization does not affect (indeed, repeating the above proof, we obtain an estimate $n \ln \ln \sqrt{n} + o(n)$that, by the properties of logarithms, asymptotically is the same), although the number of transactions decreased markedly.

Sieve only odd numbers

Since all even numbers except 2- components, we can not process any way at all even numbers and odd numbers operate only.

First, it will halve the amount of memory required. Second, it makes the algorithm will reduce the number of operations by about half.

Reducing the amount of memory consumed

Note that Eratosthenes algorithm actually operates with $n$bits of memory. Consequently, it can save significant memory consumption, not storing $n$bytes - booleans and $n$bits ie $n/8$bytes of memory.

However, this approach - **"bit compression"** - substantially complicate handling these bits. Any read or write bit will be of a few arithmetic operations, which will eventually lead to a slowdown of the algorithm.

Thus, this approach is justified only if $n$so large that $n$bytes of memory to allocate anymore. Saving memory (in $8$time), we will pay for it a substantial slowing of the algorithm.

In conclusion, it is worth noting that the language of C + + containers have already been implemented, automatically asking bit compression: vector <bool> and bitset <>. However, if speed is important, it is better to implement the compression bit manually, using bit operations - today compilers still unable to generate code fast enough.

Block sieve

Optimization of "simple screening to the root" implies that there is no need to store all the time the whole array $prime[1 \ldots n]$. To perform screening sufficient to store only the simple to the root of n, ie $prime[1 \ldots \sqrt{n}]$, the remainder of the array $prime$ to build a block by block, keeping the current time only one block.

Let s - constant that determines the size of the block, then only will $\lceil \frac{n}{s} \rceil$ block k the first block ($k = 0 \ldots \lfloor \frac{n}{s} \rfloor$) contains a number in the interval $[ks; ks + s - 1]$.Will process the blocks at a time, ie for each k of Unit will go through all the simple (as 1 before $\sqrt{n}$) and perform their screening only within the current block.Gently handle the first unit costs - first, from the simple $[1; \sqrt{n}]$ should not remove themselves, and secondly, the number 0 and 1 must be marked as not particularly simple. When processing the last block should also not forget that the last desired number n is not necessarily the end of the block.

We present the implementation of the sieve block. The program reads the number n and finds a number of simple 1 to n:

```
const int SQRT_MAXN = 100000; // корень из максимального
значения N
const int S = 10000;
bool nprime[SQRT_MAXN], bl[S];
int primes[SQRT_MAXN], cnt;

int main() {

     int n;
     cin >> n;
     int nsqrt = (int) sqrt (n + .0);
     for (int i=2; i<=nsqrt; ++i)
          if (!nprime[i]) {
               primes[cnt++] = i;
               if (i * 1ll * i <= nsqrt)
                    for (int j=i*i; j<=nsqrt; j+=i)
                         nprime[j] = true;
```

```
        }

    int result = 0;
    for (int k=0, maxk=n/S; k<=maxk; ++k) {
        memset (bl, 0, sizeof bl);
        int start = k * S;
        for (int i=0; i<cnt; ++i) {
            int start_idx = (start + primes[i] - 1) /
primes[i];
            int j = max(start_idx,2) * primes[i] - start;
            for (; j<S; j+=primes[i])
                bl[j] = true;
        }
        if (k == 0)
            bl[0] = bl[1] = true;
        for (int i=0; i<S && start+i<=n; ++i)
            if (!bl[i])
                ++result;
    }
    cout << result;

}
```

Asymptotics sieve block is the same as usual and the sieve of Eratosthenes (unless, of course, the size of $s$the blocks is not very small), but the amount of memory used will be reduced to $O(\sqrt{n} + s)$decrease and "wandering" from memory. On the other hand, for each block for each of the simple $[1; \sqrt{n}]$division is performed, it will greatly affect in a smaller unit. Consequently, the choice of the constant $s$need to keep a balance.

Experiments show that the best speed is achieved when the $s$value is about $10^4$to 10^5.

Upgrade to linear time work

Eratosthenes algorithm can be converted to a different algorithm, which is already operational in linear time - see the article "Sieve of Eratosthenes linear-time work" . (However, this algorithm has drawbacks.)